AFRL-RY-WP-TR-2008-1249

EXPLORATION OF A RESEARCH ROADMAP FOR APPLICATION DEVELOPMENT AND EXECUTION ON FIELD-PROGRAMMABLE GATE ARRAY (FPGA)-BASED SYSTEMS

Dr. Tarek El-Ghazawi, Dr. Alan D. George, Dr. Ivan Gonzalez, Dr. Herman Lam, Dr. Saumil Merchant, Dr. Proshanta Saha, Dr. Melissa Smith, Dr. Greg Stitt, Nahid Alam, Esam El-Araby, Brian Holland, and Casey Reardon

The George Washington University

OCTOBER 2008
Final Report

AIR FORCE RESEARCH LABORATORY
SENSORS DIRECTORATE
WRIGHT-PATTERSON AIR FORCE BASE, OH 45433-7320
AIR FORCE MATERIEL COMMAND
UNITED STATES AIR FORCE

NOTICE AND SIGNATURE PAGE

Using Government drawings, specifications, or other data included in this document for any purpose other than Government procurement does not in any way obligate the U.S. Government. The fact that the Government formulated or supplied the drawings, specifications, or other data does not license the holder or any other person or corporation; or convey any rights or permission to manufacture, use, or sell any patented invention that may relate to them.

This report was cleared for public release by Defense Advanced Research Projects Agency (DARPA) and is available to the general public, including foreign nationals. Copies may be obtained from the Defense Technical Information Center (DTIC) (http://www.dtic.mil).

AFRL-RY-WP-TR-2008-1249 HAS BEEN REVIEWED AND IS APPROVED FOR PUBLICATION IN ACCORDANCE WITH ASSIGNED DISTRIBUTION STATEMENT.

*//Signature//

KERRY L. HILL, Project Engineer
Advanced Sensor Components Branch
Aerospace Components Division

//Signature//

BRADLEY J. PAUL, Chief
Advanced Sensor Components Branch
Aerospace Components Division

*//Signature//

TODD A. KASTLE, Chief
Aerospace Components Division
Sensors Directorate

This report is published in the interest of scientific and technical information exchange, and its publication does not constitute the Government's approval or disapproval of its ideas or findings.

*Disseminated copies will show "//signature//" stamped or typed above the signature blocks.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YY)*	2. REPORT TYPE	3. DATES COVERED *(From - To)*
October 2008	Final	09 January 2007 – 15 June 2008

4. TITLE AND SUBTITLE
EXPLORATION OF A RESEARCH ROADMAP FOR APPLICATION DEVELOPMENT AND EXECUTION ON FIELD-PROGRAMMABLE GATE ARRAY (FPGA)-BASED SYSTEMS

5a. CONTRACT NUMBER
FA8650-07-1-7742

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER
62303E

6. AUTHOR(S)
Dr. Tarek El-Ghazawi, Dr. Ivan Gonzalez, Dr. Proshanta Saha, and Esam El-Araby (The George Washington University)
Dr. Alan George, Dr. Herman Lam, Dr. Saumil Merchant, Dr. Greg Stitt, Brian Holland, and Casey Reardon (University of Florida)
Dr. Melissa Smith and Nahid Alam (Clemson University)

5d. PROJECT NUMBER
ARPS

5e. TASK NUMBER
ND

5f. WORK UNIT NUMBER
ARPSNDBU

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
The George Washington University
Washington, DC 20052

University of Florida

Clemson University

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Air Force Research Laboratory
Sensors Directorate
Wright-Patterson Air Force Base, OH 45433-7320
Air Force Materiel Command, United States Air Force

DARPA/IPTO
3701 Fairfax Drive
Arlington, VA 22203-1714

10. SPONSORING/MONITORING AGENCY ACRONYM(S)
AFRL/RYDI

11. SPONSORING/MONITORING AGENCY REPORT NUMBER(S)
AFRL-RY-WP-TR-2008-1249

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited.

13. SUPPLEMENTARY NOTES
PAO case number DARPA 12338; Clearance Date: 4 Feb 2009. Report contains color.

14. ABSTRACT
FPGA-based reconfigurable computing (RC) technology has been widely used by the DoD community to exploit performance, power, area, cost, and versatility. Significant challenges, however, remain in application development tools and design methodologies. This study focuses on investigating key challenges and limitations of FPGA tools, and identifying and assessing potential solutions and research areas. The investigation included three major tasks: a) survey of use cases and the state of existing tools; b) characterization of limitations of existing flows, analysis of key technical challenges, and identification of potential solutions; and c) qualitative and quantitative analysis and evaluation of the solution space and projected impact. Based on the study, new concepts and methodologies based upon four phases of application development−formulation, design, translation, and execution (FDTE)−are expressed into 12 strategic research thrusts. The impact of the proposed solutions was shown to potentially revolutionize FPGA design productivity and execution efficiency.

15. SUBJECT TERMS
FPGA, reconfigurable computing (RC), design productivity, CAD tools, design flow

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT:	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON (Monitor)
a. REPORT	b. ABSTRACT	c. THIS PAGE			Kerry L. Hill
Unclassified	Unclassified	Unclassified	SAR	50	19b. TELEPHONE NUMBER *(Include Area Code)* (937) 255-4557 x3604

Standard Form 298 (Rev. 8-98)
Prescribed by ANSI Std. Z39-18

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1 Executive Summary

Advances in FPGA-based reconfigurable computing (RC) technology over the past decade have aroused significant interest throughout the community to exploit performance, power, area, cost, and versatility advantages in FPGA devices as compared to conventional microprocessor and ASIC technologies. However, significant challenges exist in application development tools and design methodologies for FPGA-based systems, a highly laborious and specialized activity, limiting the scope and widespread use of this technology. A fundamental reformation in application design methodologies and concepts to bridge the widening semantic gap between design productivity and execution efficiency is critical for the success of this technology. This research study focuses on investigating key challenges, in terms of limitations in existing FPGA tools, and identifying and developing potential research areas to address these challenges. The investigation conducted is formulated in terms of several major tasks, including: (a) survey of use cases (current and projected) and the state of existing tools; (b) characterization of limitations of existing flows, analysis of key technical challenges, and identification of potential solutions; and (c) qualitative and quantitative analysis and evaluation of the solution space and projected impact.

Several key limitations and design challenges have been identified in current FPGA design methodologies based upon survey results and FPGA tool and use case taxonomies. Chief among these limitations are: lack of Formulation concepts and tools for rapid algorithm and architecture exploration, performance prediction, and tradeoff analysis; lack of high-level, parallel Design languages enabling hardware-software co-design; unpredictable and excessively long Translation times; and lack of system-level, run-time verification and analysis tools for debug and optimization. To address these challenges, new methodologies and concepts based upon the four phases of application development, *Formulation, Design, Translation,* and *Execution* (FDTE), are critical. Twelve strategic research thrusts have been identified and defined within these four development phases that have the potential to revolutionize the application development productivity on FPGA-based systems. Many of these potential solutions are significant research challenges requiring a fundamental reformation in design methodologies and necessitating major research initiatives and investments in the field. The impacts of the proposed solutions are shown to be significant in revolutionizing FPGA design productivity and execution efficiency, which is critical for a wide variety of new and emerging uses and missions. Also, beyond the scope of RC, Formulation for RC has the potential to aid and help address some of the design productivity challenges now facing general-purpose, fixed-architecture, many-core computing technologies and therefore may potentially apply to a broadening array of computing paradigms.

As a result of this study a new research initiative, the **S**trategic **I**nfrastructure for **R**econfigurable **C**omputing **A**pplications (SIRCA), is proposed. Research results from this program would ultimately lead to a high-productivity integrated development environment (IDE) for FPGA-based applications. The proposed goals of SIRCA are:

a) Develop innovative end-to-end concepts, methods, and integrated toolsets to dramatically improve the development productivity of FPGA-based RC applications.

b) Revolutionize implementation of critical applications by broadening the use of FPGA-based RC among domain scientists and system designers for both high-performance computing (HPC) and/or embedded (HPEC) scenarios.

2 Existing Limitations and Root Causes

Throughout this study, the analysis of challenges and limitations for application development and execution on FPGA-based systems are classified based upon the four application development phases. As shown in Figure 1, these phases are *Formulation*, *Design*, *Translation*, and *Execution*. These phases encompass the overall process of producing a final Reconfigurable Computing (RC) solution, from problem formulation to execution. During Formulation, requirements are expressed using abstract modeling thereby enabling strategic exploration of the algorithm and architecture of the application. During the Design phase, details regarding the behavior and structure of the algorithm(s) and circuit architecture are specified. Mapping from the structural representation provided during the Design phase to actual resources on the target hardware producing a hardware/software solution for the platform occurs in Translation. Finally, during Execution, the run-time services are integrated to support executing, monitoring, debugging, and optimization of the full application. The baseline flow of tools for single-FPGA applications is augmented with additional tools, concepts, and support when necessary to support applications that demand the performance of multiple devices. Within each of the four FDTE phases, tools are further classified into three categories, based upon their function within that phase. As shown in Figure 2, these categories are Performance Analysis, Development, and Validation.

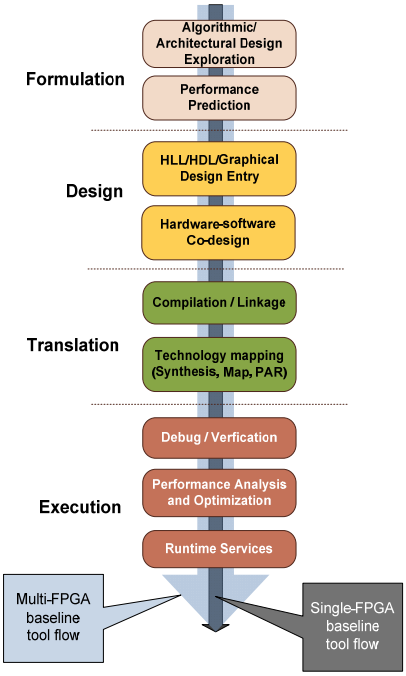


Figure 1. Application development phases

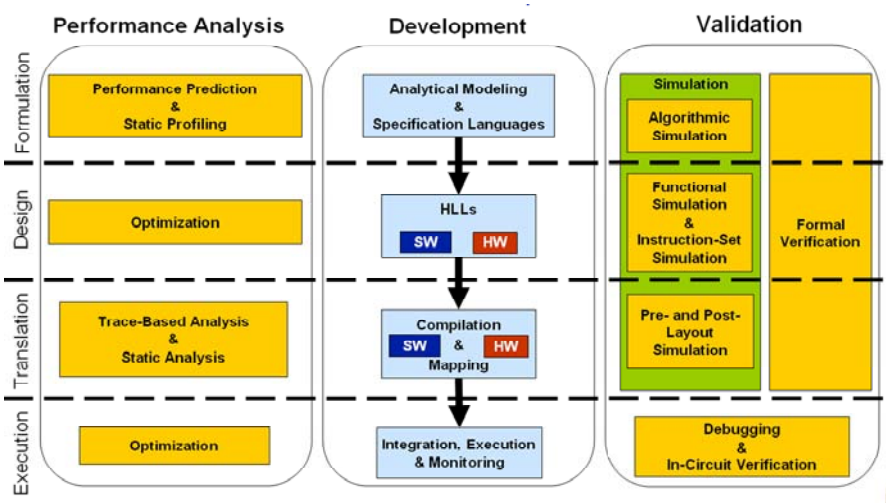


Figure 2. Tool taxonomy

A *use case* is a usage scenario that defines how the user perceives the RC system. Our use-case survey, conducted in Q3 of 2007 with approximately 30 groups using FPGA technologies, confirmed a broad range of applications, missions, and needs in the community and the specific missions and target platforms explicitly driving challenges for the tool flows. Use cases for the purpose of this study are general-purpose computing (GPC), high-performance computing (HPC) or high-performance reconfigurable computing (HPRC), and high-performance embedded computing (HPEC). In the GPC use case, the user perceives the underlying architecture as a standard von Neumann machine, such as a desktop. In the HPC/HPRC usage scenario, the user perceives the underlying machine as a parallel computer. Finally, the HPEC usage scenario uses the underlying machine as an embedded parallel computer. Figure 3 shows the usage taxonomy, where all mission scenarios are categorized into either the single- or multi-FPGA baseline flow, with additional special mission requirements as needed (e.g. for HPEC), such as real-time (RT), fault-tolerance (FT), partial-reconfiguration (PR), run-time reconfiguration, system-on-chip (SoC), and energy-efficiency considerations.

The tool flow employed during application development, while influenced by the use case, is highly dependent upon the end-user type. Application developers, for example, use a different set of tools than system designers. While application developers are satisfied using tools that abstract away the hardware, system designers require low-level tools to optimize a solution based upon the target hardware. Users are classified into two main categories, *System Designers* and *Application Designers*. Application Designers are further divided into three subcategories, *Domain Scientists*, *RC-Aware Domain Scientists*, and *RC Engineers*. The category of Domain Scientists encompasses users with little to no knowledge of the underlying FPGA-based system. The category of RC-Aware Domain Scientists is comprised of users with enough knowledge of the hardware to make explicit references to hardware-optimized functions and tools. Finally, the category of RC Engineers encompasses users knowledgeable of the entire tool chain or flow. By contrast, System Designers includes users who require advanced tools to optimize the hardware for specific target architectures. Figure 4 shows the 3-dimensional representation of the taxonomies discussed above: tools, use case, and user taxonomy.

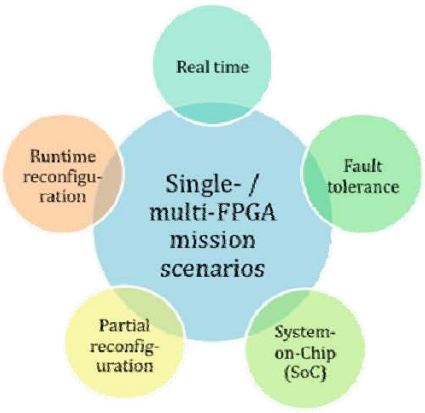


Figure 3. Use case taxonomy

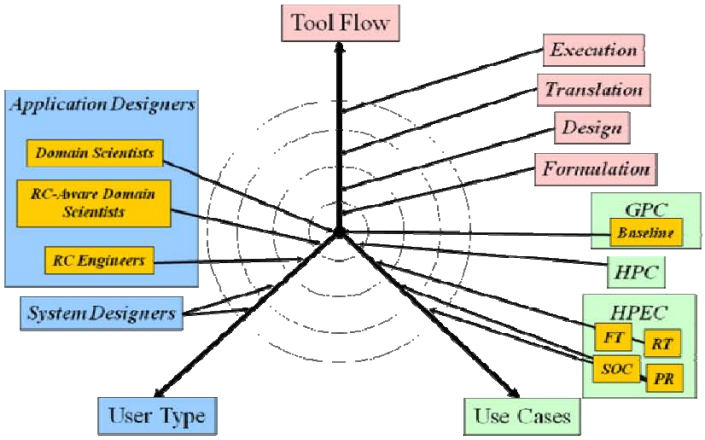


Figure 4. FPGA application development space

Analysis of our user survey results revealed the fundamental limitations that exist in the current tools and design methodologies for FPGA-based development. These limitations are described and categorized next, based upon each of the four application development phases, *Formulation*, *Design*, *Translation*, and *Execution*.

2.1 Formulation Challenges and Limitations

Formulation has been largely overlooked by existing tools and methodologies and is either skipped completely or conducted through iteratively traversing the later phases. Critical strategic design decisions are not made through algorithm exploration, architecture exploration and mapping, and numerical analysis (e.g. dynamic range vs. precision) but rather ad-hoc in the Design, Translation, and Execution phases [Edwards-97, Fornaciari-02, Gries-03]. As a result, this approach often leads to wasteful and costly design iterations resulting from poor initial design decisions. The few tools that exist for Formulation are largely inadequate and often specific to a single application domain. Furthermore, they are not integrated with other key elements of the tool flow [Densmore-06]. Without a bridge to the Design phase, any analysis and algorithm models created using such Formulation tools are discarded, necessitating a fresh start in the Design phase.

2.2 Design Challenges and Limitations

While there are many Design tools available, ranging from hardware description languages (HDLs) to high-level languages (HLLs), such tools require a significant amount of hardware knowledge to use effectively [El-Araby-07a, El-Araby-07b]. HDLs require extensive hardware expertise, thus excluding a potentially broader user base of domain scientists and programmers. Conversely, HLLs lack the ability to adequately specify the concurrency required to fully exploit the features of FPGA-based systems. The lack of integrated, system-level, hardware/software co-design tools and languages necessitates manual design partitioning and in some cases requires separate software and hardware tool flows [Planti-99, Antola-07]. Support for interoperability and portability among tools and designs is either nonexistent or limited to a small group of supported tools and platform configurations [Huang-07]. Hence, even though new HLL tools offer limited productivity improvements, more often the designs must be completely rewritten to execute on different platforms, severely limiting design/core reusability and portability. Furthermore, the scalability of designs to and across multiple FPGAs (or nodes) is largely left to the user, an arduous task at best, as there is generally no formal system-level support.

2.3 Translation Challenges and Limitations

Translation time, in particular place-and-route (PAR) time, is generally considered a major impediment to FPGA development productivity. Translation algorithms are hampered by the unpredictable and often excessively long place-and-route (PAR) times that can further vary depending on the tool chain [Li-95, Wang-96, Kannan-06]. As FPGAs become denser and more complex, these issues will continue to worsen unless new approaches are taken in the Translation phase. However, the proprietary and closed-source nature of these tools makes academic and research innovations difficult to undertake and achieve [Li-06].

2.4 Execution Challenges and Limitations

In the Execution phase, tools and methodologies for run-time support are lagging, especially for verification, analysis, debugging, and optimization at the system level [Camera-05, Tong-07]. Run-time services to support mission specific requirements such as job scheduling and completion, checkpoint and heartbeat services for fault tolerance (FT), and configuration management for run-time full/partial reconfigurations (RTR/PR) are also largely lacking. The few available run-time services are limited in functionality and are platform-specific. These problems are made worse by lack of standards for FPGA-based systems and subsystem architectures within which to integrate run-time tools [Huang-07].

2.5 Causes of the FPGA Productivity Problem

Current limitations and design challenges are symptoms of the FPGA productivity problem and not the root cause. The problem can be traced back to the incremental and evolutionary growth of tools trailing the advances in hardware capacities and innovations in FPGA devices. Figure 5 shows the evolution of FPGAs along with the concurrent evolution of design tools over the same period. FPGAs in their primordial state were mainly used for glue logic on circuit boards with RTL-level design tools. Advances in device architectures necessitated higher levels of design abstractions such as HDL, HDL+SW, and the present day HDL/HLL. Each evolution of design tools was stacked ad-hoc on top of lower layers by device vendors to merely enable use of underlying hardware resources. Tools were never designed for a broader customer base but rather were built bottom-up from Translation and Design phases outward on a need-specific basis. Technological evolution and commercial growth of this industry over the past decade has largely been aligned to support wire-line applications of the rising communications and networking industry, the largest customer of FPGA technologies. Market survivability and proprietary closed-source technology have been the chief impediments to broader growth and widespread acceptance in other fields.

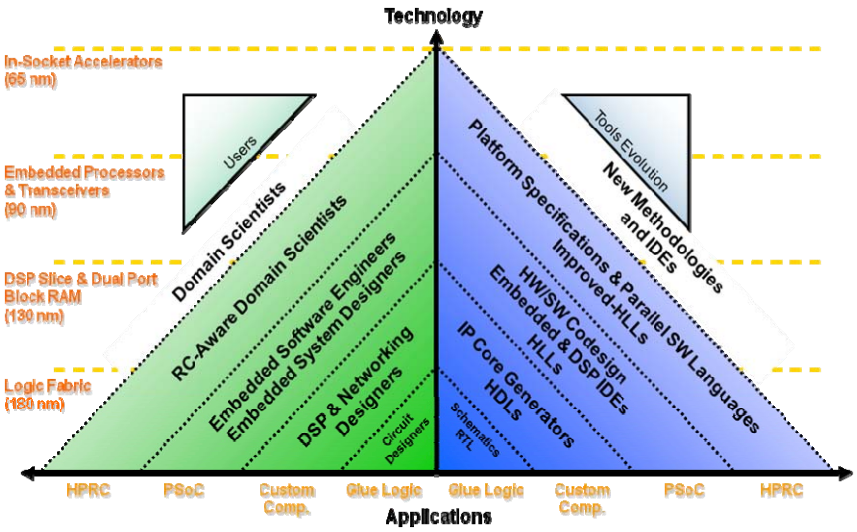


Figure 5. Technological evolution of FPGAs

3 Proposed Solutions

Computing is in the midst of an ***architecture reformation*** where unsustainable thermal and power overheads have saturated achievable clock frequencies in modern processors, ending performance growth that previously depended upon instruction-level parallelism and higher clock frequencies. The continued quest for higher performance has started a new trend focusing on thread- and task-level parallelism, leading to the emergence of homogeneous and heterogeneous multi-and many-core computing structures. Furthermore, increasingly high device design and fabrication costs will encourage use of reconfigurable computing (RC) structures ranging from gate-level, reconfigurable devices such as FPGAs [Compton-02] to interconnect-level, reconfigurable devices such as Tile-64 processor [Agarwal-07] to increase versatility and therefore market size and economy of scale of otherwise prohibitively expensive parts. Thus, diverse design paradigms and architectures from parallel and custom computing systems will eventually permeate into mainstream computing systems resulting in a technological convergence and reformation. There are many promising technologies on this new wave as illustrated in Figure 6. In this manuscript we do not distinguish between multi-core and many-core devices and use the notation MC to refer to them collectively. The two primary classes of MC architectures technology are fixed MC (FMC) devices and

reconfigurable MC (RMC) devices. FMC devices have fixed hardware structure that cannot be changed after fabrication, whereas RMC devices can change hardware structure post-fabrication to adapt to specific problem requirements. These categories can be further divided into homogeneous and heterogeneous MC devices. Homogeneous MC devices have the same type of processing core replicated across the chip, whereas heterogeneous devices have different types of processing cores optimized for specific tasks. There are many inherent challenges with FMC devices. Application-specific FMC devices (e.g. ASIC) are inflexible and expensive. Application-generic devices (e.g. Xeon and Opteron microprocessors) face power, cooling, and performance challenges. Many niches exist between these extremes (Cell, DSP, GPU, NP, etc.) that offer some flexibility within specific application domains. By contrast, RMC devices may offer the best of both worlds, with inherent advantages and tradeoffs for speed, flexibility, low-power, adaptability, and economy of scale and size [George-08, Williams-08a, Williams-08b].

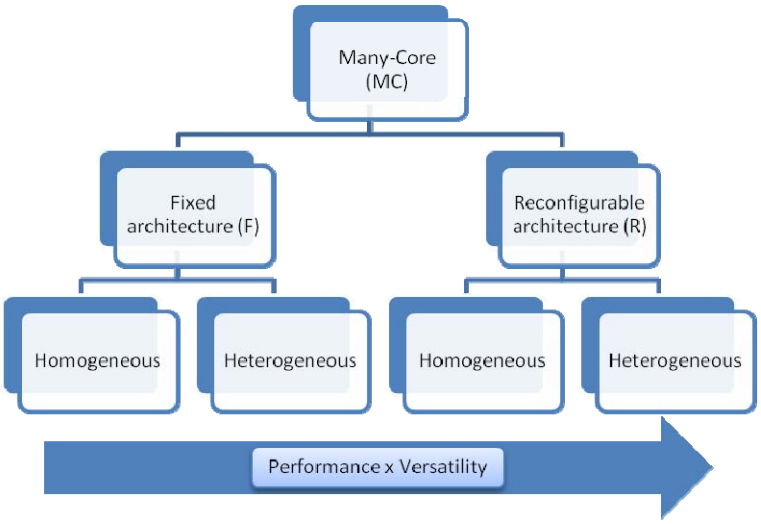


Figure 6. Evolving landscape of future computing devices

The on-going reformation in computing device architecture is leading the way to ***application reformation*** to fully exploit multiple types and layers of parallelism. The widening semantic gap between application design productivity and execution efficiency poses significant design challenges to fully exploit the potential of heterogeneous, reconfigurable, and many-core architectures. Complexity abounds and application developers must now understand and exploit parallelism. This problem is new to the mainstream computing community, but it has been fundamental and familiar to the FPGA-based RC and HPC parallel-computing community. In this application reformation, holistic concepts, methods, and tools must emerge to focus upon computational fundamentals. We also need to learn lessons from other engineering fields and build a basis for an RC engineering discipline.

Table 1. SIRCA research thrusts

	Research Thrusts	F	D	T	E
F	1. Strategic exploration	×			
	2. High-level prediction	×			
	3. Numerical analysis	×			
	4. Bridging design automation	×	×		
D	5. System-level parallel languages		×	×	
	6. HW/SW co-design methods		×	×	
	7. Reusable & portable design	×	×	×	
T	8. Translation algorithms			×	
	9. Translation target architectures			×	
E	10. Run-time debug & verification		×		×
	11. Performance analysis		×		×
	12. Run-time services				×

To accelerate and ensure the success of the application reformation, we introduce SIRCA, a Strategic Infrastructure for Reconfigurable Computing Applications. SIRCA is proposed as new set of concepts to enable development of high-productivity integrated environment for FPGA-based applications. We have identified and defined 12 strategic research thrusts to address the limitations and bridge the semantic gap between productivity and execution efficiency. These are listed in Table 1 and described below, organized across the four application development phases. Although some research thrusts span across multiple phases as indicated by × marks in Table 1, they have been grouped under their primary application development phases in our discussion below.

3.1 Formulation: Research Thrusts and Qualitative Impacts

The Formulation phase is the strategic design "playground" for managing the increasing complexity of an FPGA-based RC application. This phase involves abstract modeling of the application and strategic exploration of the algorithm and architecture of the application. Critical strategic design decisions and tradeoffs should be made in this phase, minimizing the need for "seat-of-the-pants" decisions made in the detailed Design phase. There is no coding in the traditional sense in this phase. The **research thrusts** identified for the Formulation phase are as follows.

Thrust #1: Strategic Exploration

Strategic exploration consists of high-level modeling and analysis techniques for rapid exploration and mappings of algorithms and architectures to investigate viable combinations. It does not involve any coding in the traditional sense, rather intuitive analysis and representation of the algorithm to best exploit parallelism. The representation can be graphical, textual, or hybrid with inherent constructs for expressing multiple levels and layers of parallelism such as pipelines for expressing deep parallelism and kernel replications for expressing wide parallelism. Algorithm modeling can be aided with libraries of common, predefined patterns for rapid exploration enhancing reusability. Patterns are recurring solutions for commonly faced problems, knowledge of which are often acquired through significant design experience [Dehon-04, Nagarajan-08]. Pattern-based design methodologies offer formal mechanisms for capturing and reusing these solutions or common recipes to allow novice designers to reuse the wisdom of hardware experts, enhancing their productivity. A pattern-based methodology integrated with the rest of design flow can offer a basis for strategic exploration, knowledge capture, reuse, and semi-automation to the Design phase. Figure 7 illustrates pattern-based algorithm exploration using a probably density function (PDF) estimation application as an example. Analysis of the pseudo-code for the PDF estimation algorithm reveals many commonly occurring patterns such as pipelining, loop fusion, datapath replication, and memory-dependency resolution. Algorithm modeling for this application can use pre-defined constructs for these patterns from a library for exploration and mapping to a target platform, significantly increasing user productivity. Exploration and mapping of algorithm models to target architectures can be performed using abstract hardware models of the target platforms specifying key architectural features such as I/O bandwidths, internal/external memory banks, and FPGA resources. Strategic exploration inherently involves iterative and incremental refinements to the algorithm models, mechanisms which are essential for design space exploration.

Languages for ***algorithm modeling*** can be based upon existing modeling techniques used in software engineering such as UML [UML] and ADDL [AADL], but may need to be retrofitted for hardware modeling and exploration. Alternatively, new modeling languages and techniques may be needed to satisfy the requirements of strategic exploration for RC. Figure 8 illustrates an example of PDF estimation modeling using an RC Modeling Language (RCML) under development at the Florida site of CHREC. RCML is a step towards exploring Formulation concepts and methods for FPGA application development. RCML is a hierarchical, hybrid modeling approach with built-in constructs and patterns for expressing various computation and communication mechanisms and exploring different levels and layers of parallelism.

Abstract ***hardware models*** of target architectures can also be specified in RCML for algorithm and architecture mapping and exploration. Mapped algorithm and architecture models collectively form RCML system models which can interface to other tools for prediction and design automation, as will be discussed later. Concepts of strategic exploration have also been explored in a platform-based design (PBD) approach for embedded systems design presented in [Sangiovanni-Vincentelli-07]. Platform-based design is a formal system-level development methodology based on the concepts of abstraction and iterative refinement. It is a meet-in-the-middle process, where successive refinements of specifications meet with abstractions of potential implementations. Figure 9 from [Sangiovanni-Vincentelli-07] illustrates function and architecture separation and mapping in platform-based design process. Functional representations of application specifications in platform-based design are completely independent of implementation architecture. Abstract hardware platform modules are used to represent functional description of the architecture such as processors, memories, and custom hardware. The mapping process is a design-by-refinement paradigm that binds application functionality to platforms to generate implementation.

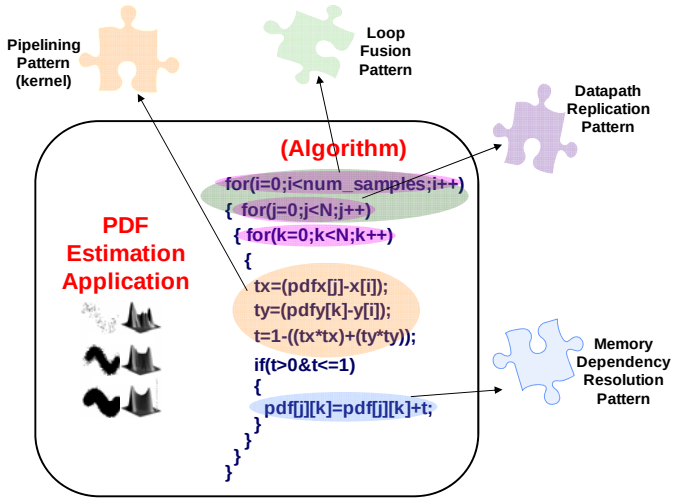


Figure 7. Pattern-based algorithm exploration

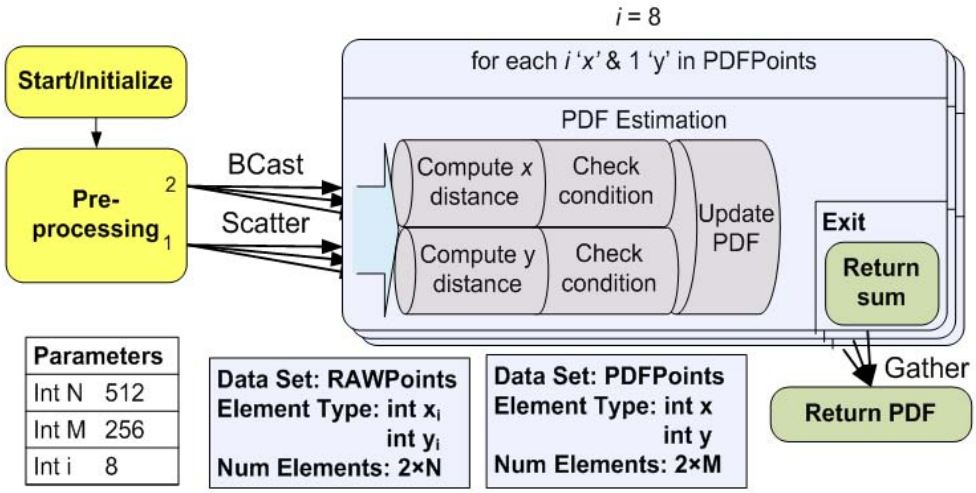


Figure 8. Example of PDF estimation modeling using RCML

Thrust #2: High-level Prediction

For mapping and exploration of algorithm models to target architectures, fast high-level performance and resource prediction methods are essential to avoid wasteful design iterations caused by poor strategic decisions producing sub-optimal designs. An application design for an FPGA-based system involves making many strategic decisions such as selecting data precision, pipeline depth, number of core replications, memory hierarchy, and data locality for available resources. Functionality and performance are inextricably linked to these decisions and dependent on correct choices for a particular target platform. These decisions are platform-specific and vary from one platform to another. High-level performance and resource prediction techniques can facilitate rapid design space exploration enabling informed strategic decisions by users. These methods can be analytical [Smith-03, Holland-07], simulative [Grobelny-07, Scrofano-07], or a combination thereof and should be seamlessly integrated in the design flow for rapid exploration. Analytical techniques typically model computational throughputs and communication overheads via representative mathematical models and are often designed for fast but low-fidelity performance and resource estimations. Simulative and hybrid analysis models are typically more detailed and designed for higher fidelity estimates and used for virtual prototyping of applications and target systems.

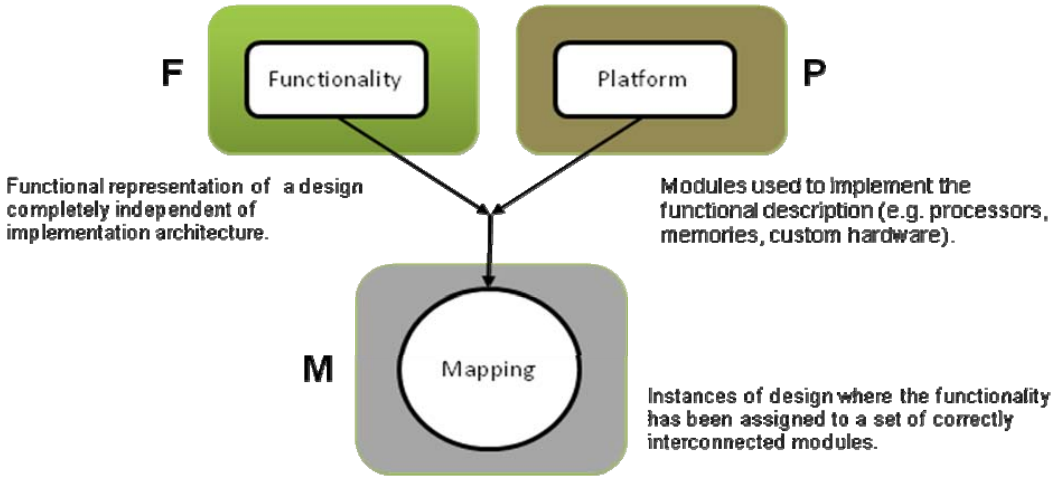


Figure 9. Function/architecture mapping in platform-based design process [Sangiovanni-Vincentelli-07]

Figure 10 shows flowcharts and frameworks for two high-level, performance prediction examples under development at the University of Florida CHREC site. The RC Amenability Test or RAT (Figure 10a) is an analytical prediction methodology that models computational throughput and communication latencies to provide quick, ball-park performance estimates for rapid design exploration [Holland-07]. RAT supports analysis of algorithms for specific target architectures by parameterizing significant elements of application behavior while characterizing the FPGA system via microbenchmarks. RAT is intended for use in strategic algorithm/architecture exploration to examine and refine the target algorithms for optimal mappings on target architectures. Figure 10b shows a diagram of an RC simulation framework [Grobelny-07]. This framework is a high-level, simulative, virtual prototyping tool for mapping arbitrary applications to targeted reconfigurable systems. The framework splits the prototyping process into two separate domains, the application and simulation domains, with six key components as shown. Under application domain, hardware and application characterizations can be performed independently and simultaneously, leading to fast and accurate performance prediction results. Generated component models and application scripts can also be reused for rapid analysis. System models driven by application scripts produce simulative performance prediction results. Several diverse FPGA platforms such as a socket-based FPGA platform (XD1000), a PCI-based FPGA platform (Nallatech cluster), and a proprietary FPGA platform (SRC-6) have been modeled using this framework.

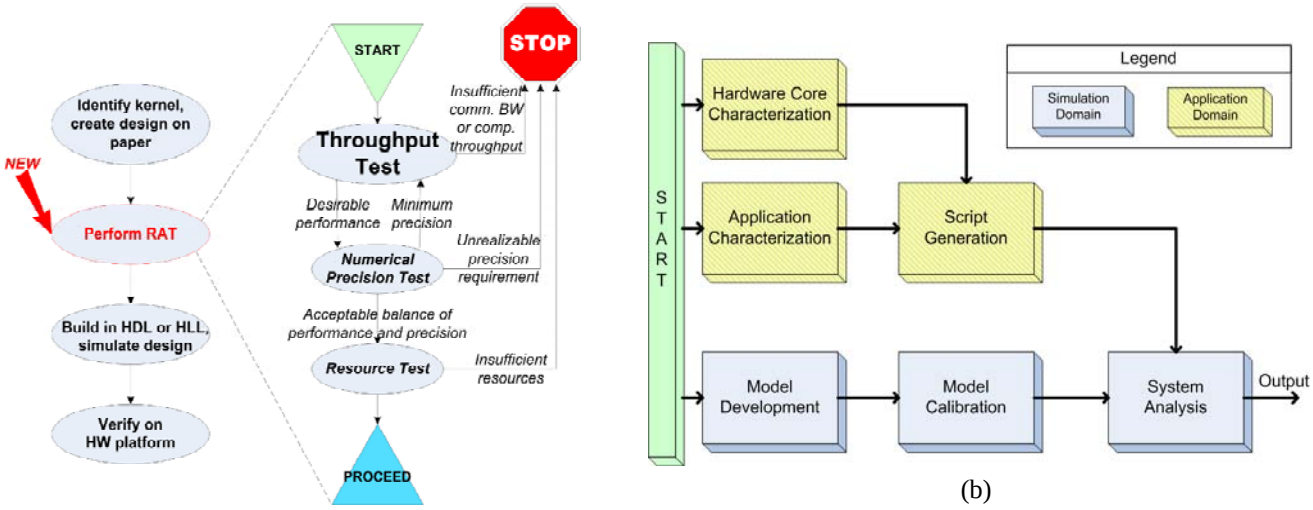


Figure 10. Flowcharts for high-level prediction methods: (a) RC Amenability Test (RAT), and (b) RC Simulation framework

Thrust #3: Numerical Analysis

The ability to use variable bit-widths for arithmetic computations gives FPGAs significant performance and resource advantages over competing computational technologies such as microprocessors and GPUs. Numerical accuracy, performance, and resource utilization are inextricably linked, and the ability to exploit this relationship is a key advantage of FPGAs. Higher precision comes at the cost of lower performance, higher resource utilization, and increased power consumption. Hence numerical analysis for optimizing bit-width selection for satisfactory implementation of target algorithms on FPGAs is essential to reap their full benefit. Numerical analysis involves selecting minimum numerical precision and data range. Data range analysis involves selecting minimum bit-widths to accommodate the dynamic range of possible numeric values, whereas precision analysis involves studying sensitivity of outputs of arithmetic components to changes in bit-widths [Lee-06]. Due to limited precision options such as *16-bit* and *32-bit integers*, and *32-bit float* and *64-bit double* floating point numbers, numerical analysis is usually circumvented by programmers and computational scientists using traditional microprocessor technologies, with the generally adopted principle being "when in doubt, use double." But with the ability to exploit variable bit-widths in FPGAs, there is a need for resurgence in concepts and integrated methods for numerical analysis in the design flow.

[Lee-06] presents an automated static approach, MiniBit, for optimizing bit-widths for fixed-point feedforward designs. The approach uses static analysis and is analytically able to guarantee maximum absolute error bounds. It uses a semi-analytical precision analysis, with analytical methods used for error models in conjunction with simulated annealing to optimize fractional bits. Figure 11 shows an overview of their approach. Although microprocessor technologies offer limited choices for precision, significant performance gains can be realized by wise selection of numerical precision for variables. Using higher precision such as *double* in place of *float* for microprocessor technologies comes at the cost of significant performance and power penalties [Buttari-07]. [Buttari-07] presents an iterative refinement methodology of using combinations of 32-bit and 64-bit floating point arithmetic precision to significantly enhance the performance of many dense and sparse linear algebra algorithms while maintaining the 64-bit accuracy of the resulting solution.

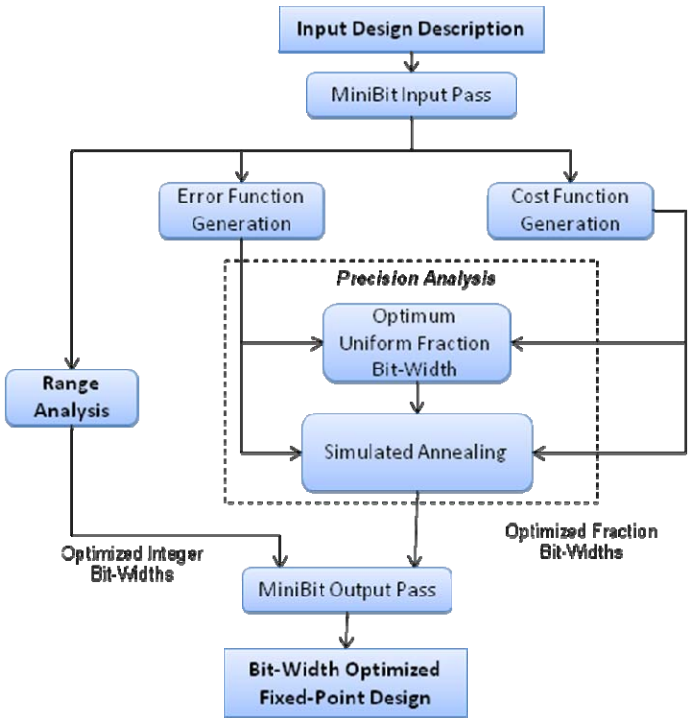


Figure 11. Overview of MiniBit bit-width optimization technique [Lee-06]

Thrust #4: Bridging Design Automation

Formulation innovations are critical in addressing the productivity gap between application design methodologies and execution efficiency on complex devices and system architectures. However, for widespread adoption of Formulation methods among users, a bridge from the Formulation phase to the Design phase is critical. Using the algorithm and

architectural models to auto-generate code templates and portions of design from pattern and core libraries can significantly enhance user productivity. This approach can form the basis for design automation where a domain scientist can model algorithms using Formulation tools and directly run automatically generated executable code on the target platform.

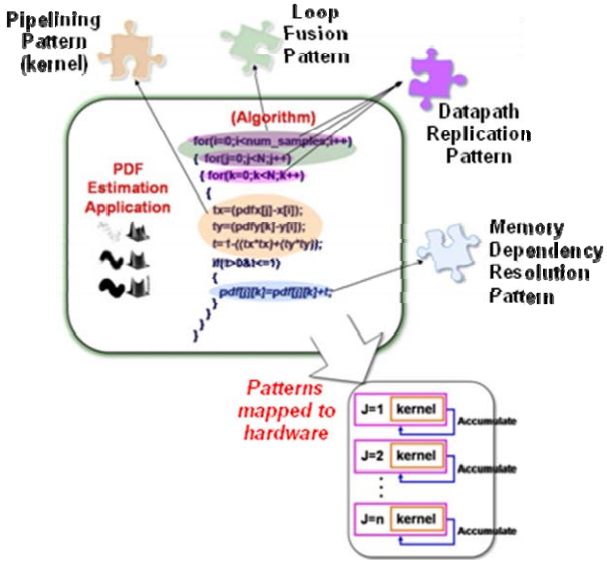


Figure 12. Pattern-based exploration used for bridging Formulation models to Design

Pattern-based algorithm exploration detailed under Thrust #1 (Strategic Exploration, Figure 7) can significantly aid in code generation. Pattern libraries used for algorithm modeling and exploration can be pre-populated with parameterized designs for each pattern, which in-turn can be used for automated code and/or template generation for bridging Formulation models to Design. This concept is illustrated in Figure 12. Although design patterns have been extensively used in software engineering, their use in hardware design has been limited [Gamma-94]. [Dehon-04] identified and classified several common hardware design patterns. Figure 13 illustrates an example time-multiplexing pattern from the paper. To our knowledge, they presented the first formal documentation template for hardware design patterns modeled similar to software engineering patterns and identified potential benefits to education and productivity via reuse, of cataloging common patterns.

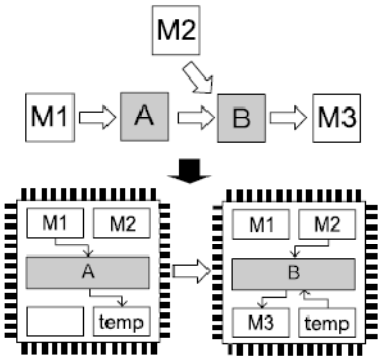


Figure 13. Time multiplexing pattern example [Dehon-04]

Qualitative Impacts of Formulation Innovations

Figure 14 illustrates an integrated Formulation methodology based upon proposed research innovations. Innovations in Formulation are expected to have significant impact on overall productivity and quality of solution. Formulation concepts with methods and tools to bridge to the Design phase, will result in a major reduction in DTE (Design, Translation, Execution) development costs, where DTE cost is a product of time spent per DTE iteration and DTE frequency (i.e. number of DTE iterations). Coding would be minimized and Design phase would be semi-automated through patterns and code

templates reducing Design time. Also, DTE frequency would be reduced since better strategic choices in Formulation mean less frequent design and re-design. This is illustrated in Figure 15. Abstract algorithm modeling and automation would result in significant utility gain (and thus productivity improvement) for non-experts. Intuitive methods and tools amenable to domain scientists and system designers will result in shorter learning curves, eliminating the need for user hardware expertise and invigorating widespread use of FPGA-based RC technology. Moreover, Formulation concepts and methods are potentially applicable to a broader range of technologies than only FPGA-based platforms, including systems featuring FMC devices and non-FPGA RMC devices.

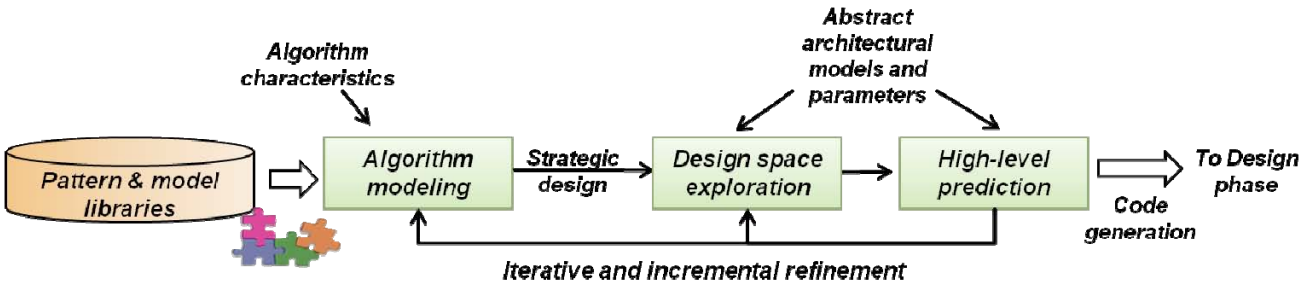


Figure 14. Formulation methodology

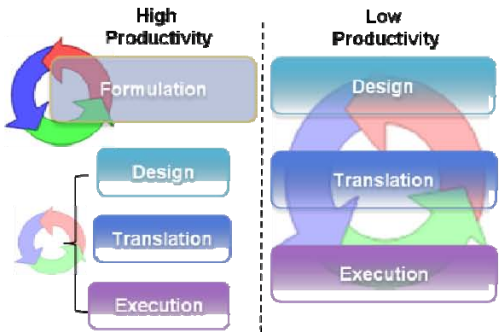


Figure 15. Impact of Formulation innovation on overall productivity

3.2 Design: Research Thrusts and Qualitative Impacts

While strategic design is performed at the Formulation phase, tactical design, which involves specification of exact behavior, is performed in the Design phase. System-level design languages, hardware/software co-design, and other design tools are necessary for design exploration, refinement and verification. Additional issues to be addressed include design reusability, interoperability, and portability. Ideally, coding is minimized through the use of design patterns and code templates resulting from the Formulation phase. The **research thrusts** identified for the Design phase are as follows.

Thrust #5: System-level Parallel Languages

RC development requires specification of computation, communication, and control for FPGAs and other components such as microprocessors, graphics processing units (GPU), and serial/parallel bus interfaces. Development, debugging, and verification of these systems involve many system-level considerations such as algorithm decomposition and architecture mappings to exploit multiple levels of parallelism, inter-device communications and control, and system-level debug and verification. Although some of these issues are explored in the Formulation phase, actual implementation occurs in the Design phase. Thus, system-level languages with constructs for expressing multiple levels and forms of parallelism are vital for efficient design implementation. Support for multi-paradigm and multi-level design abstractions in these languages can significantly improve user productivity. Multi-paradigm support enhances design productivity by simplifying design for diverse hardware devices. Hierarchical, multi-level design abstractions enable incremental and iterative design refinements for productivity versus efficiency tradeoffs. Higher abstraction levels hide the underlying hardware details and rely on automated compile and synthesis tools for efficient design implementations. Lower design abstractions expose the

underlying hardware details to the user enabling manual optimizations. While the former is more productive, the latter may achieve higher resource and execution efficiencies.

Existing design languages used for FPGAs only address a subset of the system-level design issues. Most handle hardware and software separately and provide varying degrees of design abstractions to the user. High-level languages (HLLs) such as SystemC [SystemC] and Impulse C [Pellerin-05] offer improved user productivity, but trade off execution and resource efficiency for limited portability and reusability. The Carte programming environment may achieve higher efficiency but at the cost of portability [Poznanovic-05]. By contrast, hardware description languages (HDLs) such as VHDL and Verilog often yield better performance, but come at the cost of greatly decreased design productivity. HDLs often also reduce portability and reusability. Graphical design entry tools such as Simulink [Simulink] and LabView [LabView] may also be used in conjunction with design translation tools such as Xilinx System generator [Sysgen], Altera DSP builder [DSPB], and LabView FPGA [LViewFPGA]. These tools offer better design productivity, but again are restrictive in terms of application domains, reusability, and design portability. Although these languages attempt to provide an abstraction layer to hide hardware details, significant hardware knowledge is still required for developing optimized designs. Similar design issues also face the conventional computing world with FMC devices, but are more severe with reconfigurable computing. Therefore, research innovations in FPGA Formulation and Design languages may benefit the high-performance computing and the mainstream computing technologies. By the same token, research innovations from fixed-computing programming methodologies should be leveraged for FPGA-based computing systems. Concepts and languages developed under the DARPA HPCS program such as CHAPEL, FORTRESS, and X10 could be leveraged for FPGA systems design [Ashby-07]. Design environments such as Gedae also provide potential for further research into system-level, parallel design languages [Gedae].

The programming languages for device-level design must continue to evolve, where continuing innovation is needed to raise the level of implementation abstraction above HDL to a true level associated with HLL-based design. In addition, as shown in Figure 16(a) a critical component in this infrastructure that is largely missing is system-level parallel programming languages or system coordination languages, be they graphical or textual, functional or imperative. Somewhat akin to MPI-C, MPI-Fortran, UPC, et al. in the conventional HPC community, these coordination tools would need to harness the best of the device design languages and operate above them, expressing and managing computations and data across multiple devices.

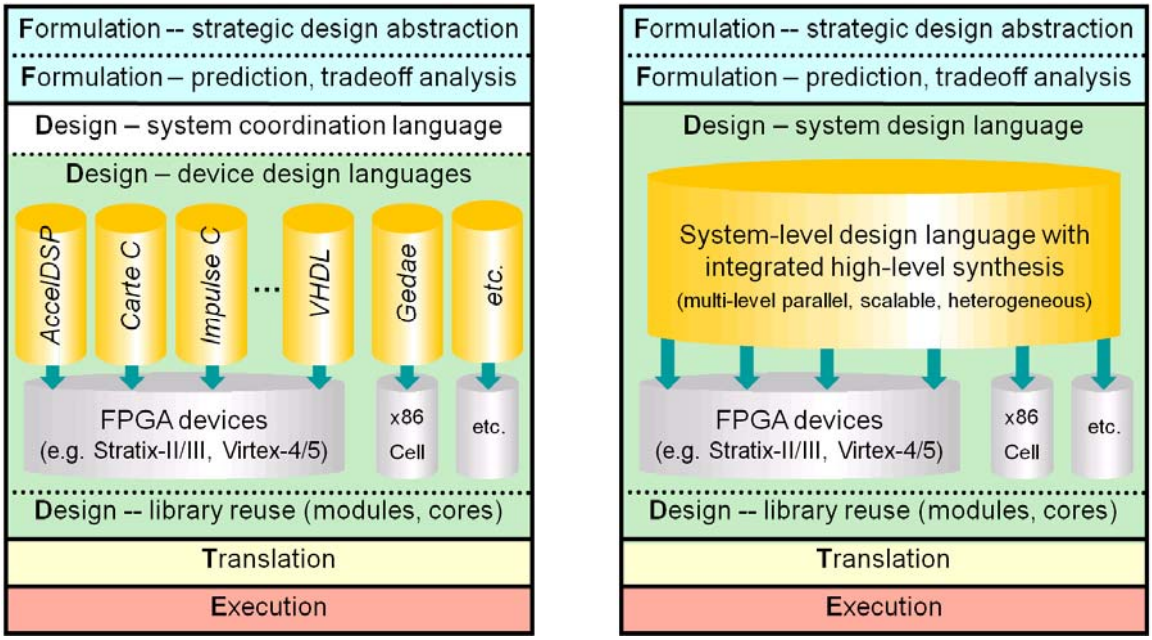


Figure 16. Major alternatives and roles for system-level parallel languages in Design phase

As illustrated in Figure 16(a), the initial emphasis of the scalable coordination language would be in coordinating mostly homogeneous computing with a set of tools and devices selected from a heterogeneous suite, where for SIRCA we emphasize

and feature the FPGA as the superior device technology for a broad range of Industrial, spacel and DoD missions. As these innovations proceed, a later emphasis would emerge, with this system-level language responsible for coordinating heterogeneous computing, involving many or all of a disparate set of devices (FPGA, CPU, GPU, Cell, etc.). Such a scalable coordination language would help bridge between the Formulation and Design layers, from strategic to tactical design and reuse. It would operate above a suite of one or more stove-pipes, pairings of design tools and devices (e.g. Impulse-C over FPGA, Gedae over Cell, OpenMP over CPU, CUDA over GPU). It and the device design languages would collectively leverage modules from expanding vendor and user libraries for code reuse and thereby exploit the productivity gains inherent with such reuse.

As an alternative to system coordination languages, abstract system-level design languages, shown in Figure 16(b), coupled with interrelated high-level synthesis tools, could potentially be used to completely separate function from architecture, thus allowing a single behavioral description to be translated on to any underlying architecture. Of course, such a language may sacrifice efficiency compared to device-specialized design languages, which are created by device experts to efficiently solve device-specific problems. However, system design languages and system coordination languages need not be mutually exclusive, potentially allowing the majority of an application to be specified with a system design languages, with performance critical regions being specified using combinations of system coordination languages and device design languages.

Thrust #6: Hardware/Software Co-design Methods

Current design flows for FPGA-based systems require separate specification and design of hardware and software. Algorithm partitioning and task scheduling are typically performed manually, a priori, before design. Due to a lack of unified hardware/software representations and co-design methodologies, system-level design, debug, and verification costs pose a major bottleneck for design productivity. Integrated and unified hardware/software co-design methods that extend to the system as a whole and not just FPGAs are essential for system-level user design perspective and user productivity. Although this problem is not unique to RC systems and has been studied extensively by embedded systems and distributed computing communities, it is much more challenging for RC systems due to multiple levels and layers of parallelism and available hardware feature diversities [Saha-07a]. But, concepts and methods developed by these communities should be leveraged and extended for FPGA-based system-level designs.

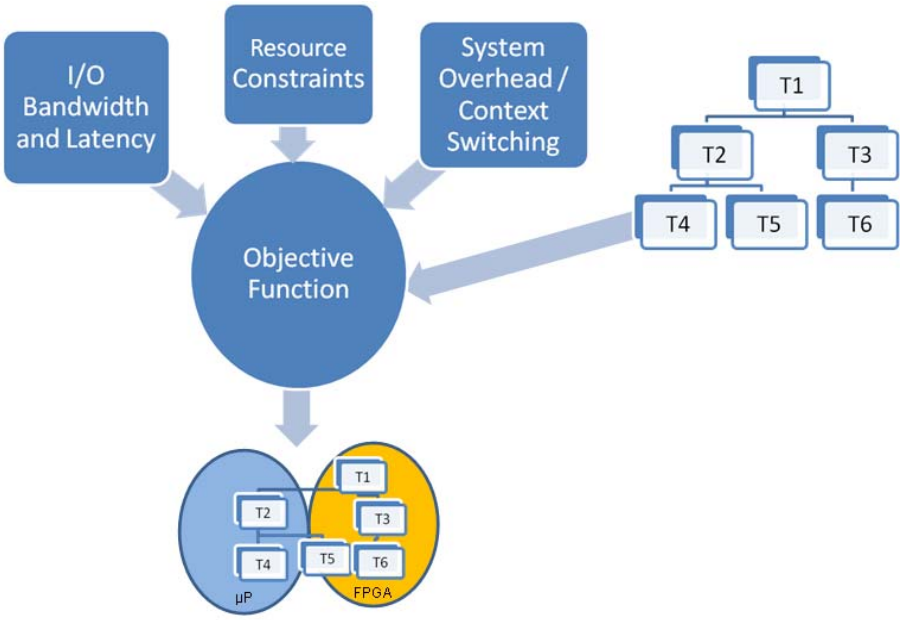


Figure 17. ReCos co-scheduling algorithm

The Polis project from UC Berkeley explored hardware/software co-design issues for real-time embedded systems [Balarin-97]. It laid the foundations of the platform-based design methodology and provided a design flow for hardware and

software partitioning and system-level co-simulation. But, it supported only a single model of computation: co-design finite-state machines (CFSMs). Metropolis was a follow-on project based on the principles of platform-based design and provided a unified and integrated environment that included a meta-model framework to implement most models of computations (MOCs) and related simulation and synthesis tools [Balarin-03]. Many concepts for hardware/software co-design explored in these projects can be leveraged for reconfigurable computing systems. [Saha-07b] proposes a co-scheduling algorithm, ReCos to address the specific hardware/software co-design concerns for reconfigurable computing systems. Figure 17 illustrates ReCos algorithm, which is under development at the George Washington University CHREC site. ReCos accepts various objective functions such as communication overhead, resource constraints, reconfiguration overhead, and application task graph as inputs. These objective functions are used to find suitable schedules for the tasks. The output of the algorithm is an optimized task graph assigned to the corresponding processing elements.

Thrust #7: Reusable and Portable Design

Built-in constructs facilitating reuse and design portability have the potential to significantly increase user productivity. Reusability in FPGA design is currently limited to reusing IP cores, but even IP reuse is restricted by lack of standards for FPGA system and sub-system interfaces to make cores portable across platforms from different vendors. Reuse has to be multi-faceted and must include reuse of cores, templates, patterns, knowledge, and run-time services. System-level frameworks and integrated environments for portable and interoperable core libraries, templates, patterns, and services for special mission scenarios such as fault tolerance, partial reconfiguration, and real-time systems are critical for fostering reuse. Pattern-based design methodologies discussed under Formulation innovations offer one such mechanism for enabling reuse of knowledge and common solution patterns. Portable core and service libraries and integrated frameworks enabling their reuse can also significantly improve user productivity. Design portability is a major problem in FPGA-based computing systems. Designs and cores written for a one platform often must be almost entirely rewritten to execute on other platforms. Different design flows and diverse hardware features and platform interfaces make achieving design portability very challenging. Interface standards for FPGA systems, sub-systems, and components can significantly enhance design portability, fostering code reuse and significantly impacting user productivity.

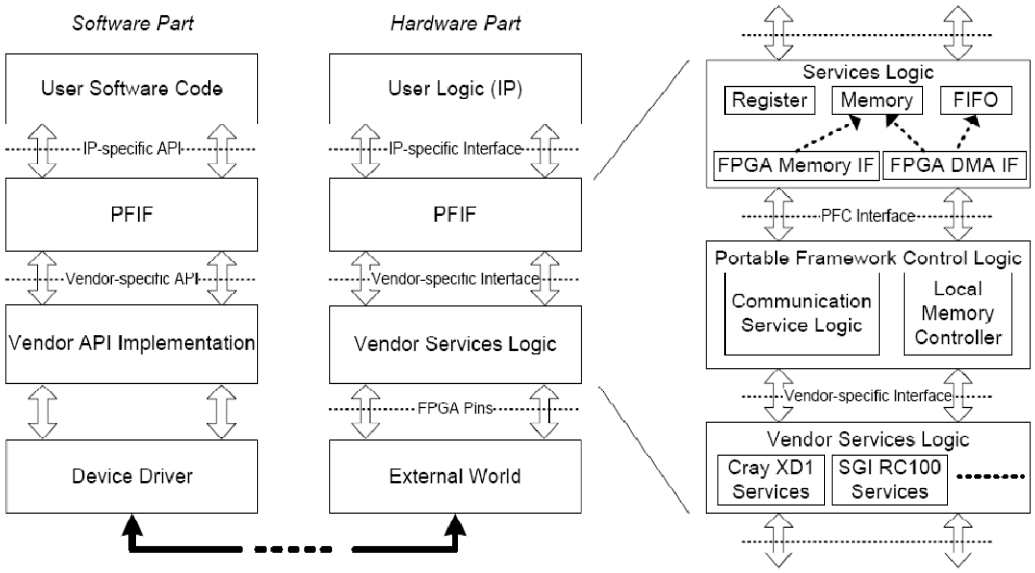


Figure 18. Portable framework interface (PFIF) [Huang-08]

Research and standardization efforts from organizations such as OpenFPGA should be supported and enhanced [OpenFPGA]. [T-GENAPI] is a technical working group at OpenFPGA for developing standard FPGA functionality application programming interfaces. Their mission is to identify and develop an application-independent, common interface for basic FPGA functionalities. [T-APPLIB] is a technical working group to develop application specific FPGA libraries. Their mission is to define the elements needed for using portable, software-callable, application-specific programming interfaces to incorporate FPGA functionality in traditional software applications. Examples of APIs may include BLAS, LaPack, ScaLapack, SciLib, PCBL, etc. [T-CORELIB] is a technical working group to develop interoperable core libraries. Their mission is to define the elements needed for increasing reusability and interoperability of the FPGA functional core

definitions that are incorporated into applications. These standardizations efforts are essential to achieve reusability and portability in FPGA application development. [Huang-08] presents a Portable Framework Interface (PFIF) for IP cores under development at the George Washington University CHREC site to address the portability issues of core libraries. PFIF framework is illustrated in Figure 18. PFIF is a layered architecture that is built on top of the FPGA platform vendor services and interfaces. At the software side, PFIF provides a set of common functions in order to allow the user to exchange data between the microprocessor and the FPGA cores. On the hardware side, the Portable Framework Control Logic (PFCL) acts as a wrapper of the vendor services and includes the vendor specific interfaces for the local memory controllers and the communication services. PFC interface to the rest of the system is standard, hence achieving the required core portability. Services logic on top of PFCL provides diverse services to the user logic that can be used by the IP, including logical memory banks, registers, FIFOs, and direct memory access (DMA) capabilities to access the microprocessor memory. A GUI supports easy customization and use of the PFIF services.

Qualitative Impacts of Design Innovations

Innovations in Design phase result in reduction in Design time due to more intuitive and high-level development methodologies and tools. Mechanisms for reuse and portability significantly improve design productivity. Better Design tools mean less frequent design and re-design, reducing not only Design time but also DTE frequency (number of iterations or cycles through the Design, Translation, Execution phases).

3.3 Translation: Research Thrusts and Qualitative Impacts

Translation tools, in particular delays in place-and-route (PAR) time, are often regarded as one of the biggest impediments to FPGA development productivity. Translation cost, which we define as the product of Translation time and Translation frequency (i.e. how often one must translate), is rapidly increasing due to larger chip densities. Translation time can be improved through innovations in Translation algorithms and through innovations in the architecture of the target devices. Translation frequency can be reduced through innovations in the other phases. The **research thrusts** identified for the Translation phase are as follows.

Thrust #8: Translation Algorithms

Innovations in Translation algorithms may significantly reduce Translation times. However, closed source and proprietary Translation tools have limited innovations by academic and other research communities. Therefore, one challenge is in fostering third-party algorithmic innovations while maintaining the intellectual property rights of vendors. Government investment can be the necessary catalyst for encouraging commercial FPGA companies to provide support for third-party innovations.

Research emphasis on high-level synthesis to produce better inputs for PAR tools can help improve Translation times [Buyukkurt-06]. Research innovations directed towards parallel Translation algorithms exploiting newer multi-core microprocessors can also significantly reduce Translation times [Haldar-00, Wrighton-03, Fobel-07]. Other potential research directions to expedite Translation times include region-based PAR methods to reduce routing complexity and improved PAR techniques to tradeoff routing effort and/or resource utilization for reduced Translation times [Mulpuri-01, Lysecky-06]. To explore new algorithms and target architectures, and make research innovations from third party researchers possible, standard abstractions for FPGA device architectures are essential. One technique is versatile place and route (VPR). As illustrated in Figure 19, the VPR technique enables interchanging algorithms and device architectures to explore new methods and better device targets [Betz-97].

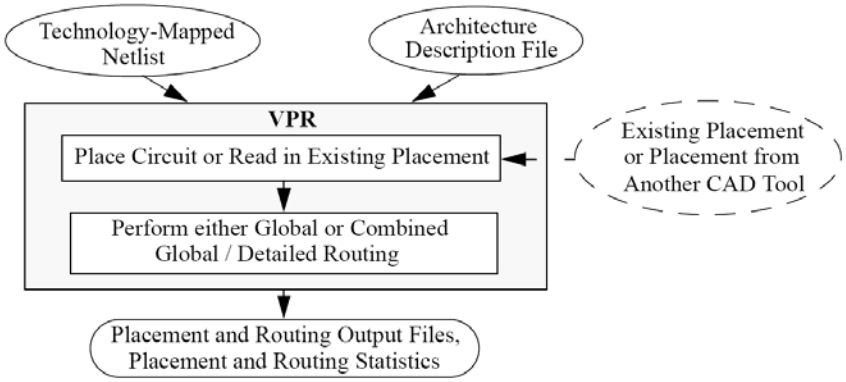
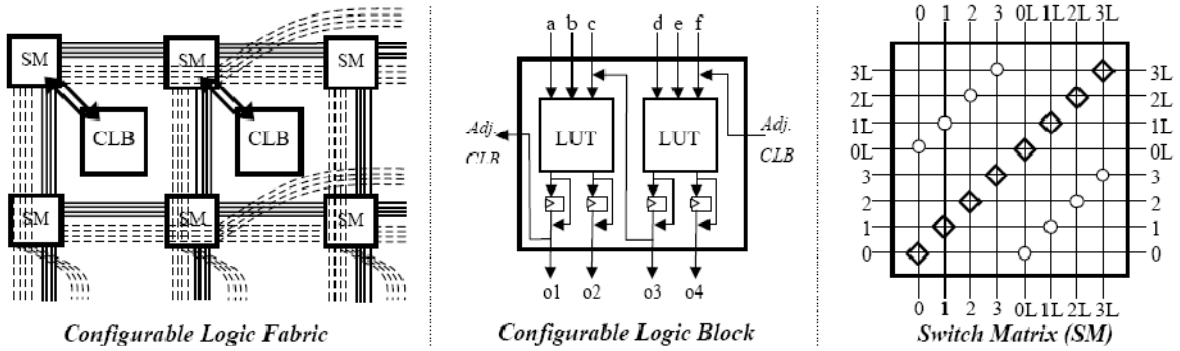


Figure 19. Example VPR technique for improving Translation algorithms [Betz-97]

Thrust #9: Translation Target Architectures

An alternative to improving Translation times is an emphasis on innovations in device architectures to provide better targets for PAR tools [Lysecky-06, Stitt-07]. The fine-grained reconfigurability and device complexity of current FPGA devices increases complexity of existing Translation algorithms. FPGAs provide a highly flexible routing structure that occupies the majority of its silicon area. Although this flexibility is needed for many circuits (e.g. glue logic), it may be overly sophisticated for many applications with data-flow architectures, which pre-dominantly use "nearest neighbor" communication patterns such as systolic arrays. Innovations in routing strategies and configurable logic structures may reduce Translation algorithm complexities thereby significantly reducing the Translation times. One possible disadvantage of Translation target architectures is that decreased routing flexibility may degrade circuit performance. However, many designers may be willing to accept a small overhead for a large reduction in Translation times.

One approach for reducing routing complexity is the use of hierarchical routing structures with tool emphasis on communication and data locality. Coarser device architectures may also help reduce PAR complexity (e.g. Mathstar FPOA). Virtual, coarse-grain architectures built on top of fine-grained FPGA devices may also provide similar functionality without the need for new silicon devices. For example, [Lysecky-04] presents an alternate routing-oriented FPGA fabric and custom routing algorithm (ROCR) for just-in-time hardware compilation (see Figure 20). Each CLB block in this fabric is connected to a single switch matrix providing all input/output connections for the CLB. The fabric has restricted routing resources as compared to traditional FPGAs and provides routing of each net only to a single pair of channels throughout the configurable logic fabric. The employed routing restrictions greatly simplify the routing algorithm enabling just-in-time compilation. Their experiments demonstrate 10× faster compilation time as compared with the well known versatile place and route (VPR) tool suite at the cost of 10% increase in additional routing resources and 32% reduction in circuit performance as compared to VPR's timing-driven router. The circuit performance when compared with VPR's routability-driven router is 10% faster using their ROCR router.



(a) (b) (c)

Figure 20. Routing oriented FPGA fabric for just-in-time compilation. (a) Configurable logic fabric, (b) configurable logic block,

and (c) Switch matrix architecture [Lysecky-04]

Qualitative Impacts of Translation Innovations

Although Translation time is generally considered a major bottleneck in development productivity, reductions in the number of iterations through the Translation phase could also have a significant impact on the overall development productivity. Innovation in Translation algorithms and device architectures will result in reduced Translation time, but not Translation frequency. Reductions in Translation frequency can be achieved via innovations in Formulation, Design, and Execution phases. Therefore, overall development productivity impact of innovations in Translation phase may be modest as compared to impact of Formulation and Design innovations. However, it should be noted that reducing Translation time may have many intangible benefits in terms of human factors. Faster Translation may impact debugging iterations and user interactions with the design and hence could result in higher productivity gains.

3.4 Execution: Research Thrusts and Qualitative Impacts

Methods and tools are necessary to provide run-time services in the Execution phase to support the debug, verification, analysis, and optimization of FPGA-based RC applications. The **research thrusts** indentified for the Execution phase are as follows.

Thrust #10: Run-time Debug and Verification

As FPGA devices, systems, and applications continue to grow in size and complexity, the difficulty and cost of debugging and verification increases as well. Unlike traditional software programs, it is not a trivial task to manually print and check the state of particular variables during execution. Instead, FPGA designers must manually implement specific circuitry for the purpose of gathering and returning debugging information. This process is very inefficient, often leading to many wasted design iterations. To maximize productivity, gathering and displaying data for the purpose of debugging and verification should be made transparent to the user, especially domain scientists who are likely to be designing at a high abstraction level. Available run-time analysis tools such as Xilinx Chipscope and Altera SignalTap are complex and require significant expertise in hardware design. Also, these do not address any system-level issues and are limited to a single FPGA. System-level, run-time debug methods and tools can significantly reduce debug and verification costs potentially eliminating wasteful debug iterations. Interface standards for FPGA systems and sub-systems with built-in debug support can help reduce verification times and increase the overall productivity.

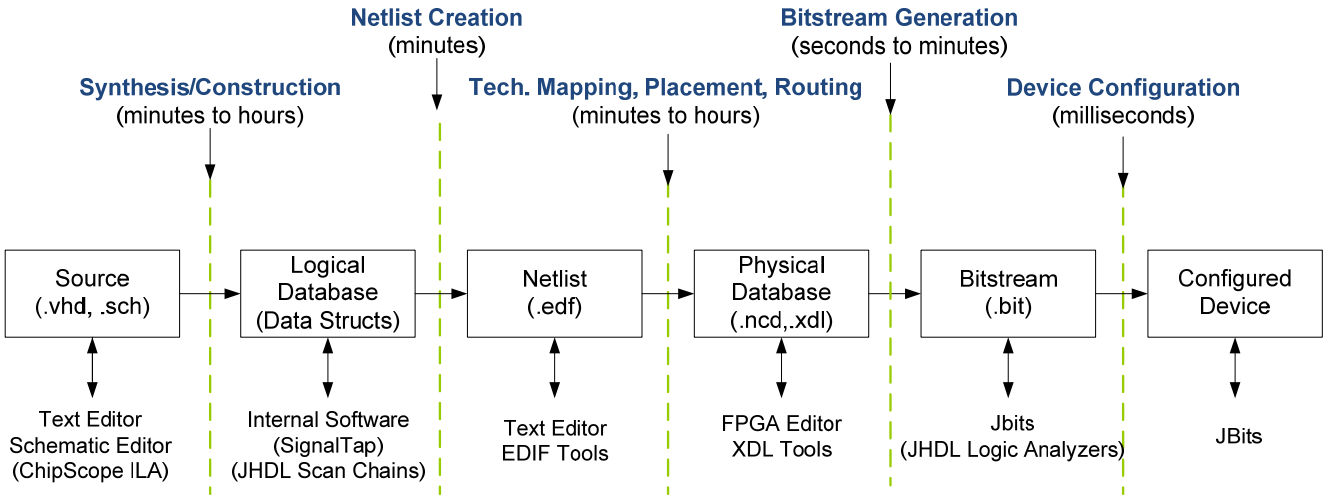


Figure 21. Possible stages for design instrumentation [Graham-01]

It is essential to instrument designs to add support for debug and verification. Instrumentation can occur at different design stages as illustrated in Figure 21 from [Graham-01]. Instrumentation can occur at the design source level (HLL/HDL), at the netlist level, or at the bitstream level. Alternately, dedicated debug circuitry can also be pre-fabricated on the FPGAs limiting the required instrumentation. [Graham-01] illustrates direct instrumentation of FPGA bitstreams to provide automated debugging capabilities. [Camera-05] demonstrates debug capabilities by inserting specialized logic before synthesis to allow for variable buffering, assertion checking, and automatic checkpointing. Source-level design instrumentation illustrated in [Koehler-07] can be used for debug and optimization of the designs. Both Xilinx and Altera provide commercial packages for automated debugging and instrumentation that allow users to insert an embedded logic analyzer (ELA) into their design. Altera's SignalTap [Altera-99] provides a GUI through which the user defines how the ELA connects to their FPGA design, which is then automatically included in the design's netlist. ChipScope [Xilinx-00] from Xilinx also allows the user to insert an ELA to their design, though ChipScope's ELA cores must be added manually to the HDL source code. An advantage of the ChipScope ELA is that modifications to the signals that interface the ELA may be made without a complete recompilation of the design. All of these approaches are single-device debug techniques. For system-level designs of FPGA-based RC systems new methods and concepts are required for automated instrumentation, data collection, visualization, and analysis for debug and verification of the system as a whole. [Tomko-00] demonstrates an environment for co-debugging of hybrid applications consisting of C/C++ software components and VHDL hardware components. The environment defines a standard interface that allows debugging data to be collected from both software and hardware components, and viewed within a VHDL simulator. Although such instrumentation techniques for debug may impose modest performance and resource penalties, the cost paid would be insignificant in comparison to improved productivity. Also the instrumentation techniques for debug and verification may equally be used for identifying design bottlenecks for critical performance and resource optimizations which may offset any penalties paid.

Thrust #11: Performance Analysis

As with debug and verification, run-time performance and resource analysis is also extremely challenging in FPGA-based HPC and HPEC systems. Increasing system-level complexities make identifying performance bottlenecks and achieving desired performance difficult due to lack of concepts and tools for run-time analysis of RC applications. Although performance analysis concepts and tools are well researched and widely available for traditional parallel computing applications, those tools fail to address the increased complexity of RC systems. The behavior of an RC application can be particularly difficult to observe and understand due to additional levels of parallelism and complex interactions between heterogeneous resources. Methods and tools tailored for run-time, post-mortem analysis of RC applications can help to quickly identify and locate critical performance and resource bottlenecks. Automated analysis and visualization of run-time performance data displayed to the user in an intuitive, high-level format can substantially improve user productivity, eliminating the need for users to manually parse large amounts of run-time performance data. Furthermore, the gathering of performance data should also be largely automated, requiring minimal additional effort from the user. Without high-level

automated tools for performance analysis, the task of run-time performance optimizations becomes tedious and unattractive to domain scientists.

[Koehler-07] presents a unified, run-time performance analysis framework for RC applications under development at the University Florida site of CHREC. This framework builds on top of and extends an existing HPC performance analysis tool, Parallel Performance Wizard (PPW), the first-ever comprehensive performance tool for UPC and SHMEM computing, also developed at Florida. Figure 22 shows various performance analysis stages used by their framework. The original application is first instrumented to modify component interfaces and add a hardware measurement module (HMM) to collect run-time data (see Figure 23). The HMM is responsible for implementing all profile, trace, and sampling capabilities, as well as packaging that data for retrieval by software. The instrumented application is then executed and run-time data is collected and analyzed for potential bottlenecks. The application is then optimized based-on the analysis results to achieve the full potential of the existing hardware resources. [Curreri-08] presents extensions of this framework to instrument and analyze high-level language FPGA applications.

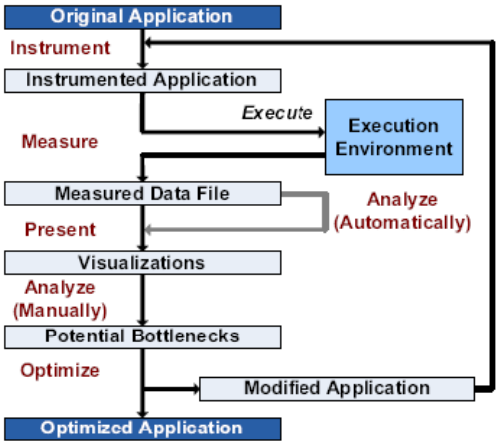


Figure 22. Stages of performance analysis [Koehler-07]

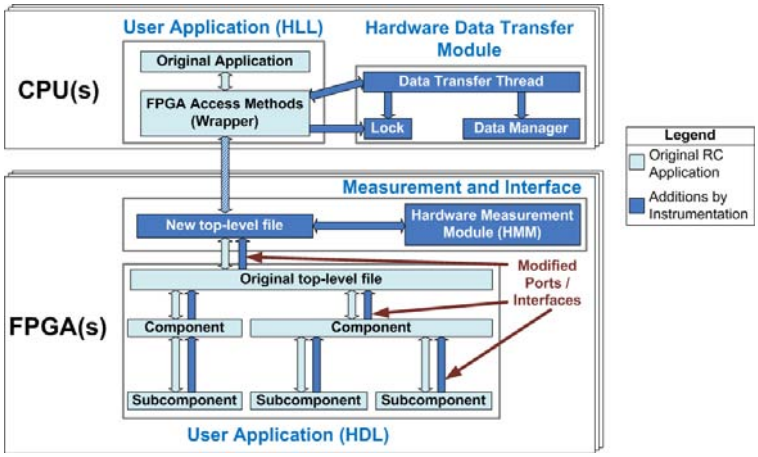


Figure 23. Source-level additions made by instrumentation [Koehler-07]

Thrust #12: Run-time Services

Run-time services can greatly simplify the FPGA development process by reducing the number of system-level issues that must be addressed by designers. By providing services at run-time that user applications can exploit, designers can obtain the desired functionality by targeting the service and rely on the execution platform to do the rest. Furthermore, as FPGAs are used in more diverse scenarios, the services needed to support their execution will become more diverse and important as well. Run-time services for FPGA systems include but are not limited to scheduling, fault tolerance management, partial-reconfiguration management, and standard operating system services such as threads and communication. Currently, these services are not provided to the user when development begins. Therefore, any FPGA design which requires the functions of such services must fully implement those functions manually. Thus, run-time services would make the implementation of such functions simpler, efficient, more reliable, and reusable.

[Lubbers-07] presents ReconOS operating system services for FPGA-based embedded real-time systems. Based on the eCos operating system, ReconOS supports both software and hardware threads in a single unified programming model. Hardware threads execute on FPGAs and interface with the operating system using the operating system interface (OSIF). OSIF provides thread synchronization services giving software-like abstraction of the reconfigurable hardware for the programmer. Figure 24 illustrates the ReconOS system architecture. [Troxel-06] presents reliable management services for a space supercomputer, the NASA Dependable Multiprocessor (DM), under development for NASA by the University of Florida and Honeywell and built using Commercial-Off-The-Shelf (COTS) technologies. The DM platform is composed of a collection of COTS data processors augmented with COTS FPGAs and radiation hardened system controllers to interface with the spacecraft control computer. A middleware layer provides various components such as reliable messaging, fault-tolerance managers and agents, and job managers and agents as illustrated in Figure 25. These components provide reliable

run-time communication, fault recovery, and job scheduling and management services for the applications executing on data processors and FPGAs.

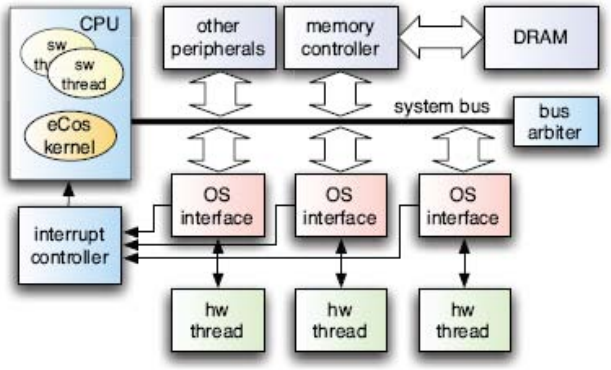


Figure 24. ReconOS System architecture [Lubbers-07]

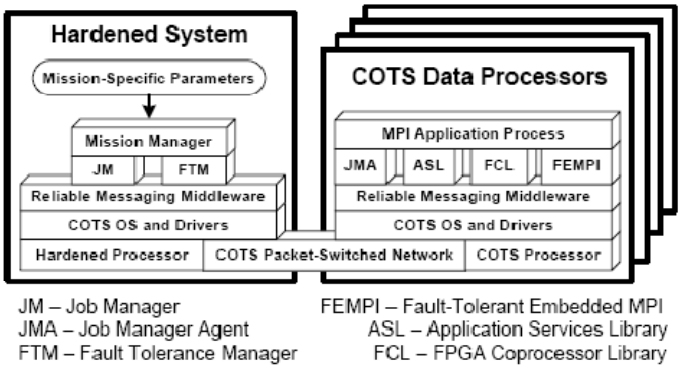


Figure 25. DM Middleware Architecture [Troxel-06a]

Qualitative Impacts of Execution Innovations

Superior Execution tools can offer significant development cost savings, and reduction in debug time and DTE frequency. Critical optimizations and bottleneck elimination can also result in significant utility gains, consequently increasing productivity.

4 Quantitative Productivity Analysis with Existing Tools

In the previous section, we presented the *qualitative* impact of the proposed innovations on productivity for each strategic research direction. Next, a two-prong approach to *quantitatively* predict the impact of the innovations on productivity will be described. A quantitative analysis of the impact based on *existing tools* will be discussed in this section, followed by a quantitative analysis of the impact based on the *projected future* of the FDTE innovations in Section 5.

Both quantitative impact analyses are based on the following ***RC Productivity Model***. Productivity, as defined by the DARPA HPCS program [Kepner-04a], is the ratio of utility over development cost:

$$\psi = \frac{U}{C} \tag{1}$$

Utility U is a function of a set of weighted attributes a_i used to measure the usefulness of a design. Examples of utility attributes include speed, throughput, power efficiency, and area efficiency. The magnitude of utility can be measured as the length of the utility vector in a multi-dimensional utility space:

$$U = length\left(\overline{U}\right) \tag{2}$$

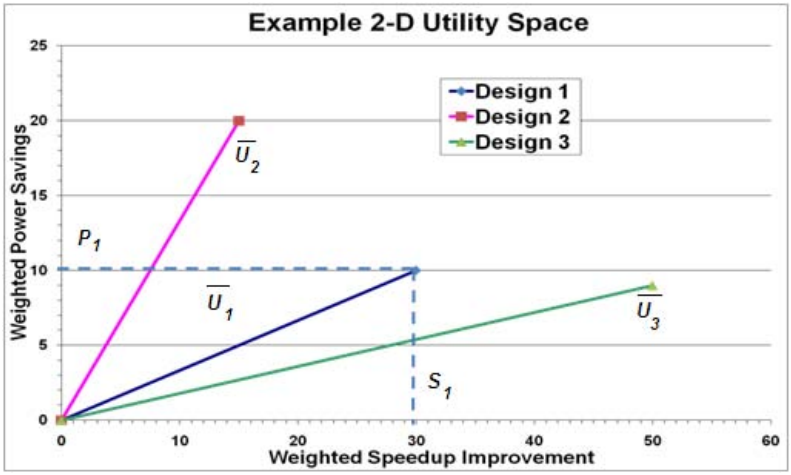


Figure 26. Example 2-dimensional utility space

Figure 26 shows an example of a 2-dimensional utility space, using utility attributes *weighted speedup S* and *power savings P*. For example, the utility for Design 1 is:

$$U_1 = length\left(\overline{U_1}\right) = \left|\overline{U_1}\right| = \sqrt{S_1^2 + P_1^2} \tag{3}$$

The development cost *C* is the work required to achieve that utility and is typically measured as development time (e.g. man-hours) or dollars. The cost for an RC application prototype is modeled as the summation of the costs for the four phases of the FDTE model: Formulation (C_F), Design (C_D), Translation (C_T), and Execution (C_E). Each application may require the development of multiple (*k*) prototypes. Each prototype is represented as a design iteration *i*.

$$C = \sum_{i=1}^{k} \left(C_F + C_D + C_T + C_E\right)_i \tag{4}$$

The productivity number is often more meaningful when compared with a reference. Hence, our analysis will mainly calculate relative productivity gains given by the ratio of two productivities. Typically, ψ_2 is used as a reference:

$$Productivity\,Gain = \frac{\psi_1}{\psi_2} = \frac{U_1}{C_1} \div \frac{U_2}{C_2} \tag{5}$$

Again, a two-prong approach to *quantitatively* validate the impact of the innovations on productivity will be described. The quantitative analysis of the impact based on *existing tools* will follow in this section. A quantitative analysis of the impact based on the *projected future* of the FDTE innovations will be discussed in Section 5.

The analysis of current tools allows an understanding of the potential of the proposed FDTE Methodology. Current tools are primarily Design-centric, where it is usually assumed that the user has the necessary hardware development skills for required for RC platforms.

The comparative study conducted in [El-Araby-07a, El-Araby-07b] considered some of the available tools; see results shown in Figure 27. The goal of that study was to measure (or estimate) the productivity of the selected tools taking into consideration the acquisition time (i.e. the time required to learn the tool), the development time, and the utility. The utility was measured based on how good the tool is in reaching the target (or optimal) requirements, e.g. frequency and area utilization. The table in Figure 27 shows the average results obtained from different groups of users developing a set of IP cores using the selected tools. The graph in Figure 27 shows the relative productivity space of the tools based on the total cost. The target (or optimal) utility was set to 200 MHz for frequency and 0% for area utilization.

Previous results in Figure 27 show the productivity of each individual tool. However, it is possible, as shown in Figure 28, to classify these tools following the FDTE Methodology. Although all of the tools are Design-centric, some of them (SysGen and RC Toolbox) present environments that enable a higher design abstraction. These tools were selected to represent the current status of Formulation tools, although their formulation capabilities are limited (or they lack formulation

capabilities). We classified these tools as Formulation-Like Flow tools. The remaining tools were grouped as Design-Centric Flow.

	Tool	Acquisition Time (Days)	Development Time (Hours)	Frequency (MHz)	Area (% Utilization)
1	C	2	3	0	100
2	Impulse-C	7	6	125	21.25
3	Handel-C	8	7.5	200	20.25
4	Carte-C	6	6.5	100	19.5
5	Mitrion-C	10	10.75	100	22.75
6	SysGen	8	9	200	19
7	RC Toolbox	9	9	200	19
8	HDLs	15	15.25	200	16.75

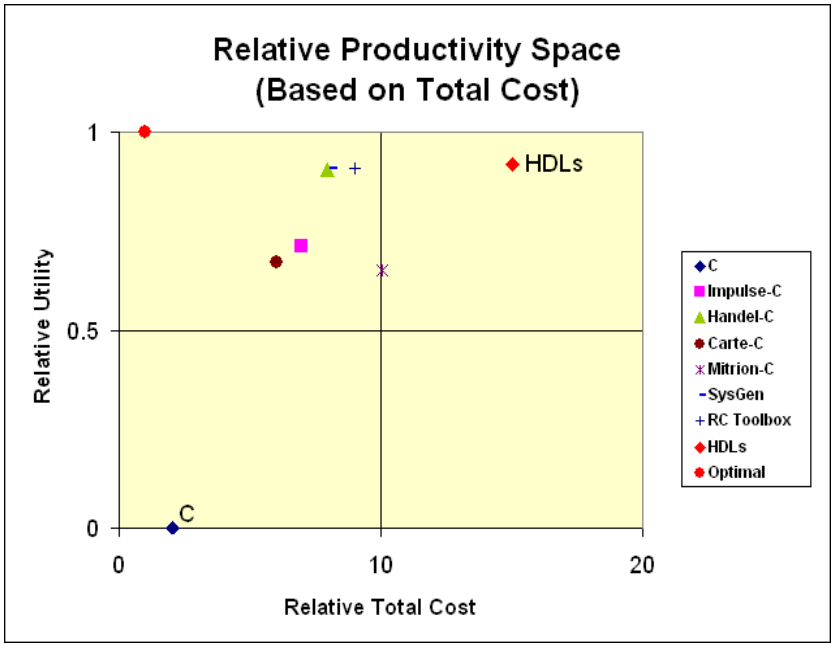


Figure 27. Relative productivity and results from selected tools

In addition to the selected tools, we needed to identify Translation and Execution tools. However, Translation and Execution are not design-entry phases; they are optimization and debug phases. Translation tools are involved in the design process in a transparent way, like for example, when users need to synthesize or execute P&R tools. There are two kinds of Translation tools. First, there are synthesis and Place & Route tools, most of which are automatic tools that allow the user to improve the results by increasing the effort or changing some execution parameters. Second, there are manual place and route tools like PlanAhead, where the user needs a deeper knowledge of the reconfigurable hardware. We observed the fact that, to use these tools, it is required to execute the Formulation and/or Design phases first. Therefore, to evaluate these tools, a Translation/Execution-Centric Flow will be considered. With these assumptions, we define:

- Translation/Execution Utility:
  - 20% Improvement with respect to Design (e.g. Area, Frequency) based upon vendor literature
- Translation/Execution Cost

- Acquisition $time_{Translation}$ = Acquisition $time_{Formulation}$ + Acquisition $time_{Design}$
- Development $time_{Translation}$ = $T_{Translation}$ + MIN(Development $time_{Formulation}$, Development $time_{Design}$)

Based upon classification of tools and previous definitions, Figure 29 summarizes the estimated values for each type of FDTE tools showing the execution costs in form of the acquisition and the development time as well as the utility numbers for the frequency and the utilized area. Considering this data, the productivity results of each development phase can be represented as shown in Figure 30. The Formulation-Like design environments that enable a higher design abstraction show the best productivity gain because they achieve the best results in terms of frequency and utilized area in the shortest amount of time. The lowest productivity gain is achieved by the Translation/Execution-Centric Flow because the development costs are significantly higher. Development tools at that level provide capability to significantly influence placement and routing of circuit components which results in long acquisition and especially long development times.

The observation based on the study of current tools indicates that Formulation as an entry point has a potential for higher productivity impact. Spending time and effort in Formulation can result in a higher productivity than Design or Translation/Execution. Furthermore, Formulation is performed on a more abstract level and does not require detailed knowledge about the targeted RC system. This higher level of abstraction can potentially expand the user base to domain scientists and programmers significantly improving their design productivity. However, the current Formulation tools are still immature because they are close to Design-Centric than Formulation. Another important need is the seamless integration of the different tools (e.g. IDE with multiple entry points for different users).

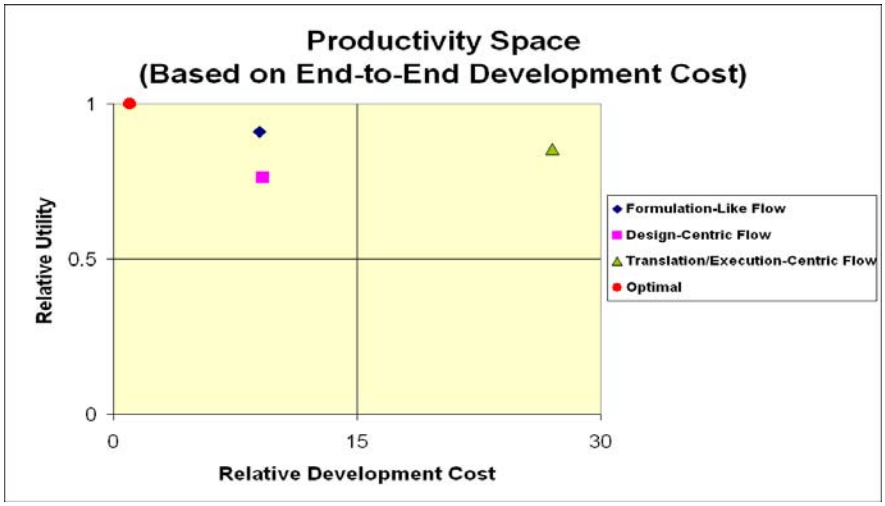
Finally, based on the results obtained from current tools, we can identify the cost implications of the FDTE methodology. While emphasizing that Formulation may have maximum potential impact on productivity, it entails significant tool development complexity and the associated costs for the vendor, as shown in Figure 31.

	Tool
Formulation-Like Flow	SysGen, RC Toolbox
Design-Centric Flow	Impulse-C, Handel-C, Carte-C, Mitrion-C, HDLs
Translation/Execution-Centric Flow	XST, Synplify, PlanAhead, Quartus, Leonardo, ChipScope, etc.

Figure 28. FDTE classification of selected tools

	Flow	Acquisition Time (Days)	Development Time (Hours)	Frequency (MHz)	Area (% Utilization)
1	**Formulation-Like Flow**	8.5	9	200	19
2	**Design-Centric Flow**	9.2	9.2	145	20.1
3	**Translation/Execution-Centric Flow**	17.7	27	174	16.08
4	**Optimal**	1	1	200	0

Figure 29. Relative productivity and results from selected tools



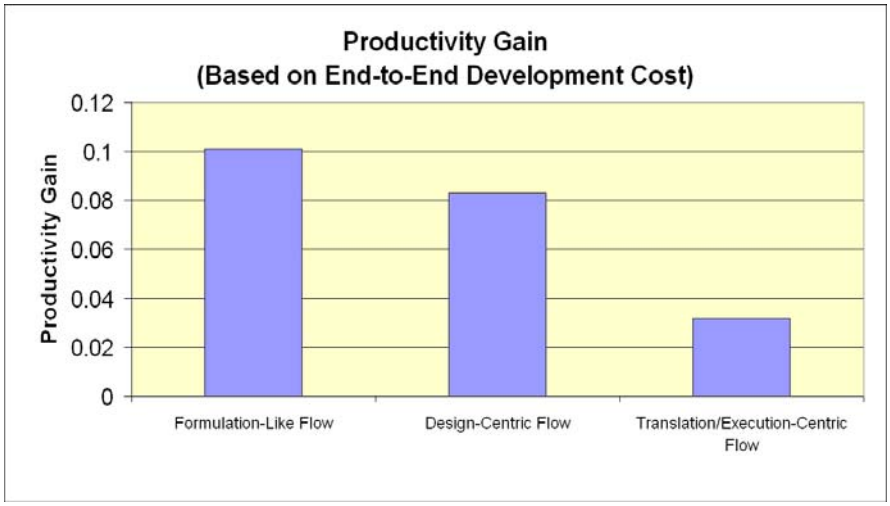


Figure 30. Productivity space and gain from selected tools

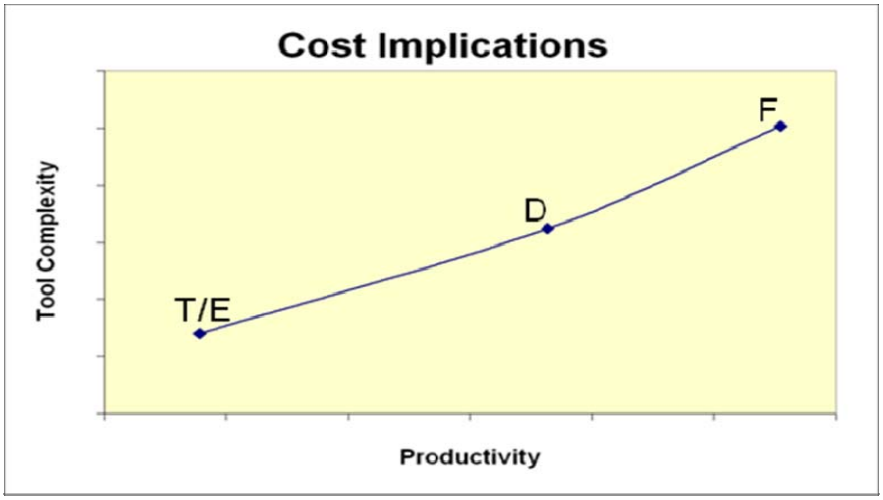


Figure 31. FDTE cost implications

5 Quantitative Productivity Analysis with Future FDTE Innovations

Before presenting the details of the analysis, we first describe our general approach and define the key assumptions. This information provides the background for the projections to be estimated.

Approach. Productivity impact is determined by many factors, interrelated in a complex manner. To keep our analyses tractable and to gain a clear understanding of the analyses, our approach is to break the complex problem into its component factors in order to isolate and focus on the impact of each factor. Our analysis is set up as follows. There are five analyses. For the first four, the impact from the increase in utility *U* is assumed to be minimal, while we analyze the productivity effect based only on the change in cost *C*. These four analyses are as follows:

1 Assume that there are innovations ***only*** in the ***Formulation*** phase (as a result of SIRCA), with no innovation in the other three phases (holding them constant for simplicity and clarity), and determine the end-to-end development cost and the impact on productivity.

2 Similar to #1, but with innovations ***only*** in the ***Design*** phase, determine productivity impact.

3 Similar to #1, but with innovations ***only*** in the ***Translation*** phase, determine productivity impact.

4 Similar to #1, but with innovations ***only*** in the ***Execution*** phase, determine productivity impact.

For Analysis #5, using a different scenario, we assume the impact from the increase in cost *C* is minimal and we analyze the productivity effect based only on the change in utility *U*.

For all five analyses, the cost of each phase is the product of time spent in that phase and the number of iterations through that phase (i.e. frequency). For example, the Translation cost is $C_T = f_T \times t_T$

Two projections were made to compare to a defined baseline: conservative, and optimistic. The conservative projection represents expected improvements that should certainly result from SIRCA. Optimistic estimates represent ambitious but attainable productivity gains.

Table 2. Baseline development costs

Development Phase	Development cost (hrs)	
	Baseline	*Rationale*
Learning Curve	400 (s/c)	*HDL learning curve*
Formulation	40 (1w)	*Current Formulation methods*
Design	480 (12w)	*Estimated from literature*
Translation	200 (5w)	*Estimated from experience*
Execution	280 (7w)	*Estimated from literature*
Total	400 (10w) + 1000 (25w) = 1400 (35w)	

Baseline assumptions. The five analyses assume a baseline development cost as a reference for relative productivity gains. The assumptions for the baseline costs are as follows. The percent breakdown for the baseline development time per phase was inferred from reported percentages of five HPC and HPEC software projects; Hawk, Saturn V, SAGE, NTDS, and Gemini [Kepner-04b, Kendall-05]. Note that the percentages were adjusted to reflect the development of an FPGA-based RC application (e.g. longer Translation time):

Formulation = 5%, Design = 50%, Translation = 20%, and Execution = 25%

The total baseline development time is assumed to be 1000 hrs. Also, the analyses assume the user is a domain scientist knowledgeable only in sequential programming. The baseline code is assumed to be developed in HDL, with a learning curve cost of 400 hrs (approximate time for an intensive one-semester university course (s/c)). Based on these assumptions, Table 2 shows the resulting numbers for the baseline development costs (in units of hours and weeks).

5.1 Impact of Formulation Innovations

For this analysis, the assumption is that we are considering innovations ***only*** in the ***Formulation*** phase (as a result of SIRCA), and there are no innovations in the other three phases (holding them constant). Also, we assume that there is no

increase in utility *U*, but there will be a decrease in cost *C*. This approach represents the scenario of a user who is capable of getting the most utility out of a design. Thus, new innovations in Formulation tools will not help him/her obtain more utility (e.g., speedup). However, it can help him/her produce a design faster (decrease in cost *C*).

Detailed results are shown in Table 3 and corresponding data is plotted in Figure 32. Cost improvements due to innovations in each FDTE phase are estimated based on empirical knowledge. Note that the rationale for cost reduction for each row of the table is also given. As is evident, if we spend more time and effort in the Formulation phase (80 hours or 2 weeks for both the conservative and optimistic cases), it results in significant reduction of cost in the other three phases. Innovations in Formulation, with methods and tools to bridge to the Design phase, will result in a reduction in the learning curve and Design time. Coding would be minimized and Design phase would be semi-automated through design patterns and code templates from Formulation. Also, the number of prototype designs required and DTE frequency would be reduced, since better strategic choices in Formulation means less frequent design and re-design.

Table 3. Impact of Formulation innovations

Development Phase	**Development Cost (hrs)**			
	Baseline	***Conserv.***	***Optim.***	***Rationale***
Learning Curve	400 (s/c)	40 (1w)	8 (1d)	*Reduction in time required to learn high-level tools & methodologies*
Formulation	40 (1w)	80 (2w)	80 (2w)	*New FDTE will spend more time in F to save time in DTE*
Design	480 (12w)	160 (4w)	20 (½ w)	*Reduction in Design time (templates) & frequency (strategic exploration)*
Translation	200 (5w)	80 (2w)	20 (½ w)	*Reduction in Translation frequency, optimistically approaching one step*
Execution	280 (7w)	160 (4w)	20 (½ w)	*Lower frequency (strategic exploration) & less run-time optimization & bugs*
End-to-End Dev Cost	1400	520	148	
Productivity Gain		1400/520 = 2.7	1400/148 = 9.5 *(from 35 weeks to only 3.7 weeks)*	

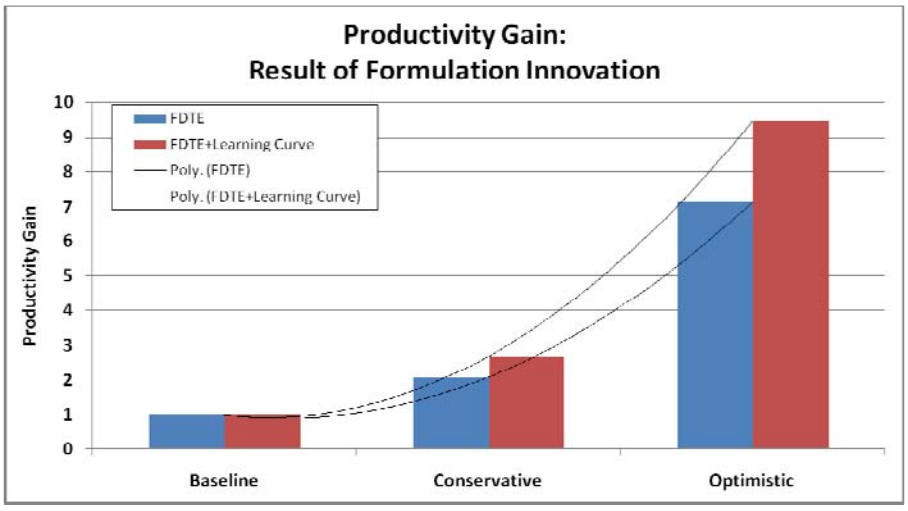


Figure 32. Productivity gain from Formulation innovations

5.2 Impact of Design Innovations

Similar to the previous subsection, the assumption here is that we are considering innovations ***only*** in the ***Design*** phase (as a result of SIRCA), and there are no innovations in the other three phases. Also, we assume that there is no increase in

utility, but there will be a decrease in cost. The detailed results are shown in Table 4 and the corresponding data is plotted in Figure 33. The rationale for the cost reduction for each row of the data is also given.

In general, innovations in the Design phase result in many of the same benefits as Formulation. Improved Design methodologies lead to considerable DTE cost reduction, including a reduction in Design time, which is a result of intuitive, high-level design tools. There is also a reduction in DTE frequency (e.g. less need for debugging and optimization) due to better concepts and tools. However, due to lack of strategic exploration in Formulation, it does not reduce prototype iterations caused by poor strategic design decisions.

Table 4. Impact of Design innovations

Development Phase	Development Cost (hrs)			
	Baseline	*Conserv.*	*Optim.*	*Rationale*
Learning Curve	400 (s/c)	80 (2w)	16 (2d)	*Reduction in time required to learn high-level tools & methodologies*
Formulation	40 (1w)	40 (1w)	40 (1w)	*No impact*
Design	480 (12w)	240 (6w)	80 (2w)	*Reduction in Design time due to intuitive languages & methodologies*
Translation	200 (5w)	120 (3w)	40 (1w)	*Fewer design flaws lead to reduction in Translation frequency*
Execution	280 (7w)	200 (5w)	40 (1w)	*Fewer design flaws lead to reduction in Execution(e.g. debugging frequency)*
End-to-End Dev Cost	1400	680	216	
Productivity Gain		1400/680 = 2.1	1400/216 = 6.5 *(from 35 weeks to only 5.4 weeks)*	

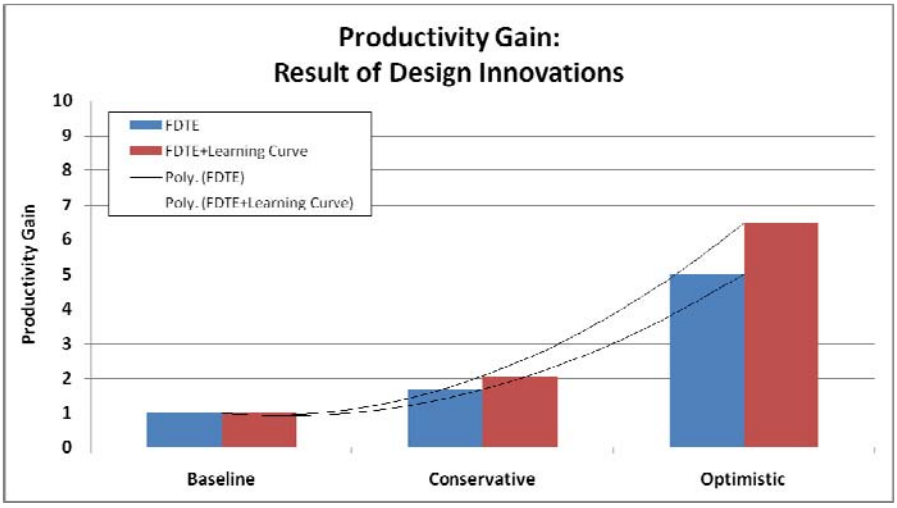


Figure 33. Productivity gain from Design innovations

5.3 Impact of Translation Innovations

As before, we are considering innovations ***only*** in the ***Translation*** phase, thus we assume that there are no innovations in the other three phases. We also assume that there will be a slight increase in utility and a decrease in cost. The detailed results are shown in Table 5 and the corresponding data is plotted in Figure 34.

Although Translation time (T time) is often considered one of the biggest impediments to FPGA-based development productivity, it is shown here that the impact of innovations in reducing T time may not affect overall productivity significantly. The reason is that innovations in Translation do not affect F, D, or E costs, nor the T frequency, which is more critical than just reducing T time. As shown in the earlier analyses, T frequency reductions can be achieved via innovations in the F, D, and E phases. Note that unlike the previous analyses, we also assume a 10% (conservative) and a 30% (optimistic) gain in utility, assuming better Translation algorithms and devices will improve utility. Even with this assumption, the productivity gain is modest.

It should be noted that our analyses does not account for any intangible benefits of reductions in Translation times. Benefits of faster Translation may impact debugging iterations and user interactions with the design and could result in higher productivity gains. Although reductions in Translation frequency may have higher impact on overall productivity, Translation time improvements are vitally important and innovations here must not be ignored.

Table 5. Impact of Translation innovations

Development Phase	Development Cost (hrs) *Baseline*	*Conserv.*	*Optim.*	*Rationale*
Learning Curve	400 (s/c)	400 (s/c)	400 (s/c)	*No impact*
Formulation	40 (1w)	40 (1w)	40 (1w)	*No impact*
Design	480 (12w)	480 (12w)	480 (12w)	*No impact*
Translation	200 (5w)	40 (1w)	8 (1d)	*Significant reductions in Translation time, not frequency*
Execution	280 (7w)	280 (7w)	280 (7w)	*No impact*
End-to-End Dev Cost	1400	1240	1208	
Utility Gain		10%	30%	*Better resource utilization and clock frequencies due to innovated tools*
Productivity Gain		1.1 × (1400/1240) = 1.2	1.3 × (1400/1208) = 1.5	

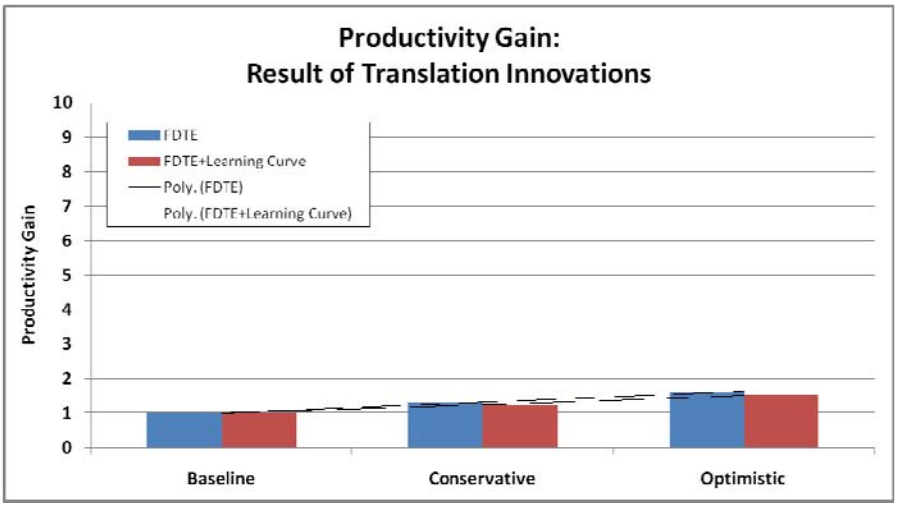


Figure 34. Productivity gain from Translation innovations

5.4 Impact of Execution Innovations

As before, we are considering innovations ***only*** in the ***Execution*** phase, thus we assume that there are no innovations in the other three phases. We also assume that there will be a slight increase in utility and a decrease in cost. The detailed results are shown in Table 6 and the corresponding data is plotted in Figure 35. The rationale for the cost reduction for each row of the data is also given.

Innovations in Execution methods and tools will reduce debug, verification, and optimization costs in the DTE phases. They will also reduce the DTE frequency to minimize the number of debugging and optimization cycles. The learning curve is reduced since run-time analysis tools will become more intuitive to the user. As in the case of Translation, this analysis assumes a 10% (conservative) and a 30% (optimistic) gain in utility because innovations in Execution methods and tools will help identify bottlenecks and critical design optimizations that may increase utility (e.g. speedup).

Table 6. Impact of Execution innovations

Development Phase	Development Cost (hrs) *Baseline*	*Conserv.*	*Optim.*	*Rationale*
Learning Curve	400 (s/c)	360 (9w)	320 (8w)	*Easier learning curve for run-time analysis & optimization tools*
Formulation	40 (1w)	40 (1w)	40 (1w)	*No impact*
Design	480 (12w)	360 (9w)	280 (7w)	*Reduction in Design frequency and debugging time*
Translation	200 (5w)	120 (3w)	80 (2w)	*Fewer debugging cycles leads to reduction in Translation frequency*
Execution	280 (7w)	160 (4w)	40 (1w)	*Better tools quickly identify bottlenecks and bugs*
End-to-End Dev Cost	1400	1040	760	
Utility Gain		10%	30%	*Critical design optimization and bottleneck identification*
Productivity Gain		1.1×(1400/1040) = 1.5	1.3×(1400/760) = 2.4	

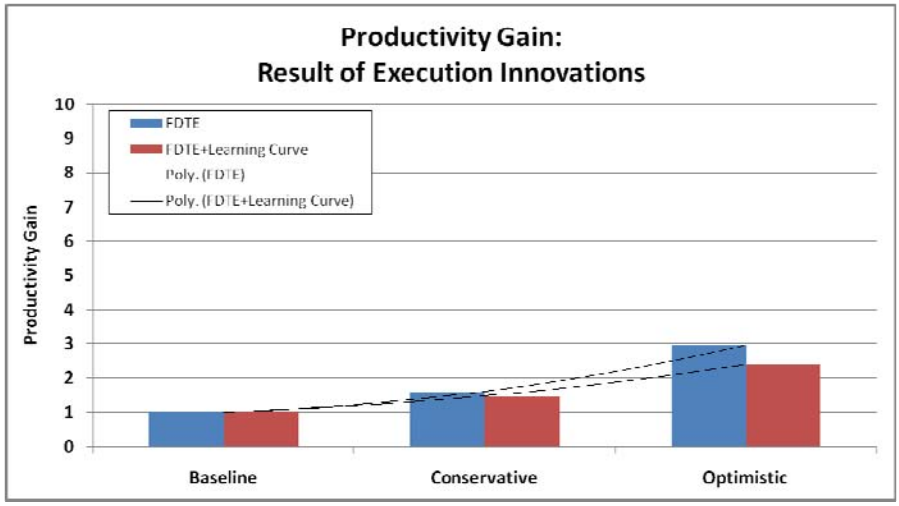


Figure 35. Productivity gain from Execution innovations

5.5 Impact Summary Based upon Cost, Holding Utility Constant

Based on the four analyses described in the previous sections, we can rank the research directions in terms of the projected impact of the innovations for the respective development phases. The projected productivity impact for each phase is plotted in Figure 36, using the baseline development cost as a reference.

1. Formulation innovations have the potential for maximum impact on productivity. Also, Formulation innovations have the potential to dramatically increase the user base for FPGAs, cascading a multiplicative effect for productivity. Finally, although strategic exploration concepts are central to RC, they are also potentially applicable to the GPC world.
2. In general, innovations in the Design phase result in many of the same benefits as Formulation. Design innovations by themselves increase overall productivity. However, Formulation and Design innovations together will likely generate maximum impact.
3. Execution innovations are important for providing critical run-time information. They enable the reduction in optimization, debug, and verification costs.
4. Improvement in Translation time is beneficial, but that alone is less impactful on overall productivity.

Our analyses have focused on the isolation of the impact for innovations in each development phase. It should be clear that the cumulative impact for all four phases will yield maximum benefit. Unfortunately, since the cumulative productivity impact is a complex function of the individual impacts, we cannot simply add them together. However, we can make some meaningful observations. Recall that innovations in all four phases can combine to drive DTE down, whereas none of the other innovations will drive F lower. Thus, theoretically the maximum achievable productivity, considering development time alone, will be limited by the Formulation cost and learning curve. Thus, as DTE costs approach 0, the development cost is equal to the learning curve cost plus the Formulation cost. As shown by the dashed horizontal lines in Figure 36, for the conservative case, the gain approaches 1400/120 = 12, and for the optimistic case 1400/88 = 16. Note that these theoretical maximums are based on the numbers in Table 3 and will vary depending on the particular case study. Further, productivity is unlike speedup; a 12× reduction in cost (*C*) would be truly outstanding (e.g. man-year effort reduced to man-month).

Finally, all the analyses thus far considered only the improvement in the cost part of the productivity equation. We have held the utility part of the equation constant. It should be clear that the productivity benefits would be even more pronounced when utility improvement from using FPGA-based RC is included, as will be shown in the next analysis.

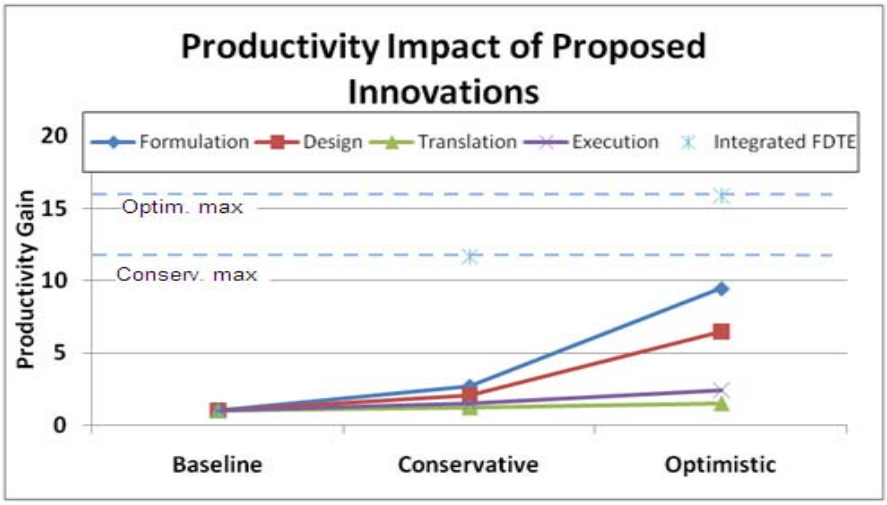


Figure 36. Productivity impact of proposed innovations

5.6 Projected Impact based upon Utility, Holding Cost Constant

This analysis evaluates the productivity impact resulting from the utility of FPGA-based RC as compared to fixed computing devices. With systems featuring fixed-architecture, multi-core and many-core devices, many of the development problems faced by RC are now also faced for GPC, although more severe in RC due to benefits of variable architecture as opposed to fixed with FMC devices. Thus, this analysis assumes comparable development costs, and evaluates the impact based on the utility of RC. Of course, we do not expect RC development cost to be equal to fixed-computing costs. But, in a "SIRCA-enabled" world, we believe we can get close in the future, especially as development in the FMC world gets harder as parallelism expands. Additional assumptions for this analysis are follows:

- FMC device baseline is modern microprocessors
- Conservative: 10× speedup, 10× power savings, 2× dev cost with RC vs. fixed-computing
- Optimistic: 50× speedup, 20× power savings, 1.2× dev cost with RC vs. fixed-computing

(Note: This range of utility gain is common in today's literature [Shackleford-01, Masuno-07, Merchant-07, Williams-08b].)

Based on these assumptions, a two-dimension utility space (power savings P and speedup S) is shown in Figure 37, comparing the utility of the three cases. Recall from Section 4 that the utility for a
Design 1 is:

$$U_1 = length\left(\overline{U_1}\right) = \left|\overline{U_1}\right| = \sqrt{S_1^2 + P_1^2}$$

Shown in Figure 38 is the Productivity Gain as compared to the baseline fixed-computing system with FMC microprocessors.

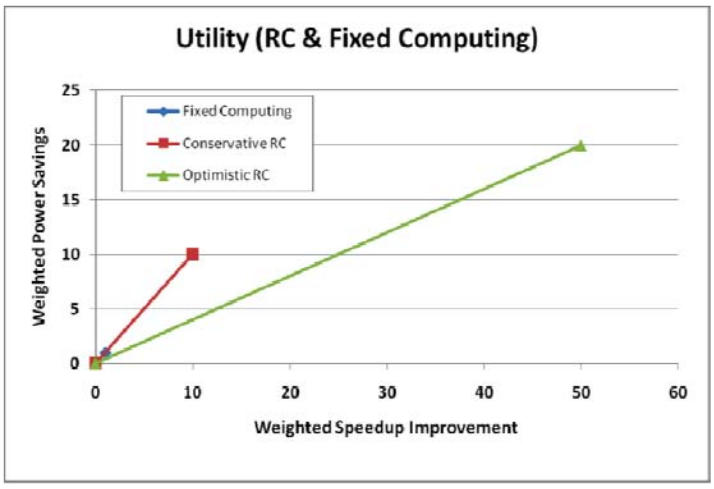


Figure 37. Utility comparison

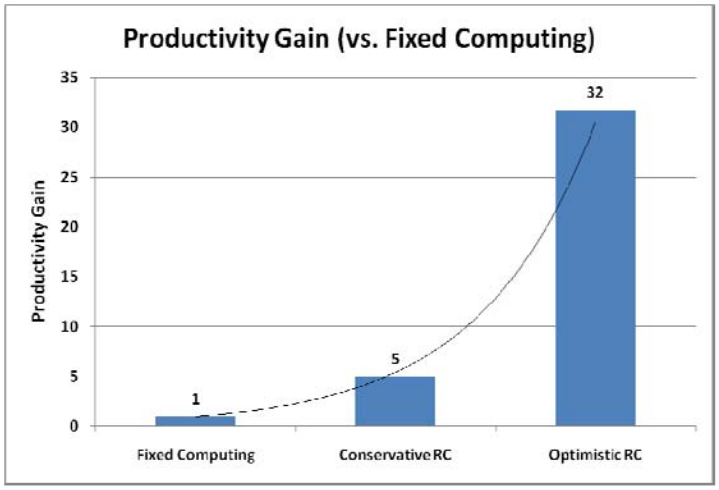


Figure 38. Productivity gain comparison

In summary, it has been shown in this analysis that the productivity benefit is even more pronounced when utility improvement from using FPGA-based RC is included. Furthermore, FDTE innovations will make RC tools amenable to domain scientists. System-level, hierarchical tools will abstract the underlying hardware details. Automation will facilitate targeting FMC and RMC devices with significantly less effort, making performance, power, versatility, and cost advantages of FPGAs available to a larger user base.

6 SIRCA: A Proposed New Research Program

Shown in Figure 39 is the summary roadmap for the proposed SIRCA program. To recap, the program goals for SIRCA are:

a) Develop innovative end-to-end concepts, methods, and integrated toolsets to dramatically improve the development productivity of FPGA-based RC applications.

b) Revolutionize implementation of critical DoD applications by broadening the use of FPGA-based RC among domain scientists and system designers for both high-performance computing (HPC) and/or embedded (HPEC) scenarios.

Also summarized are the motivations for a program and the technical areas of particular interests. These are the 12 research thrusts identified and discussed in Section 3 of this report. It should be noted that the concept diagram for the FDTE model on the bottom left of the figure emphasizes one of the key conclusions from this study. That is, more time and effort spent in the Formulation phase will result in significant reduction of cost in the other three phases and thus reduction in the overall development cost. Formulation innovations have the highest potential impact on productivity. Also, Formulation innovations have the potential to dramatically increase the user base for FPGAs, thus cascading a multiplicative effect on development productivity.

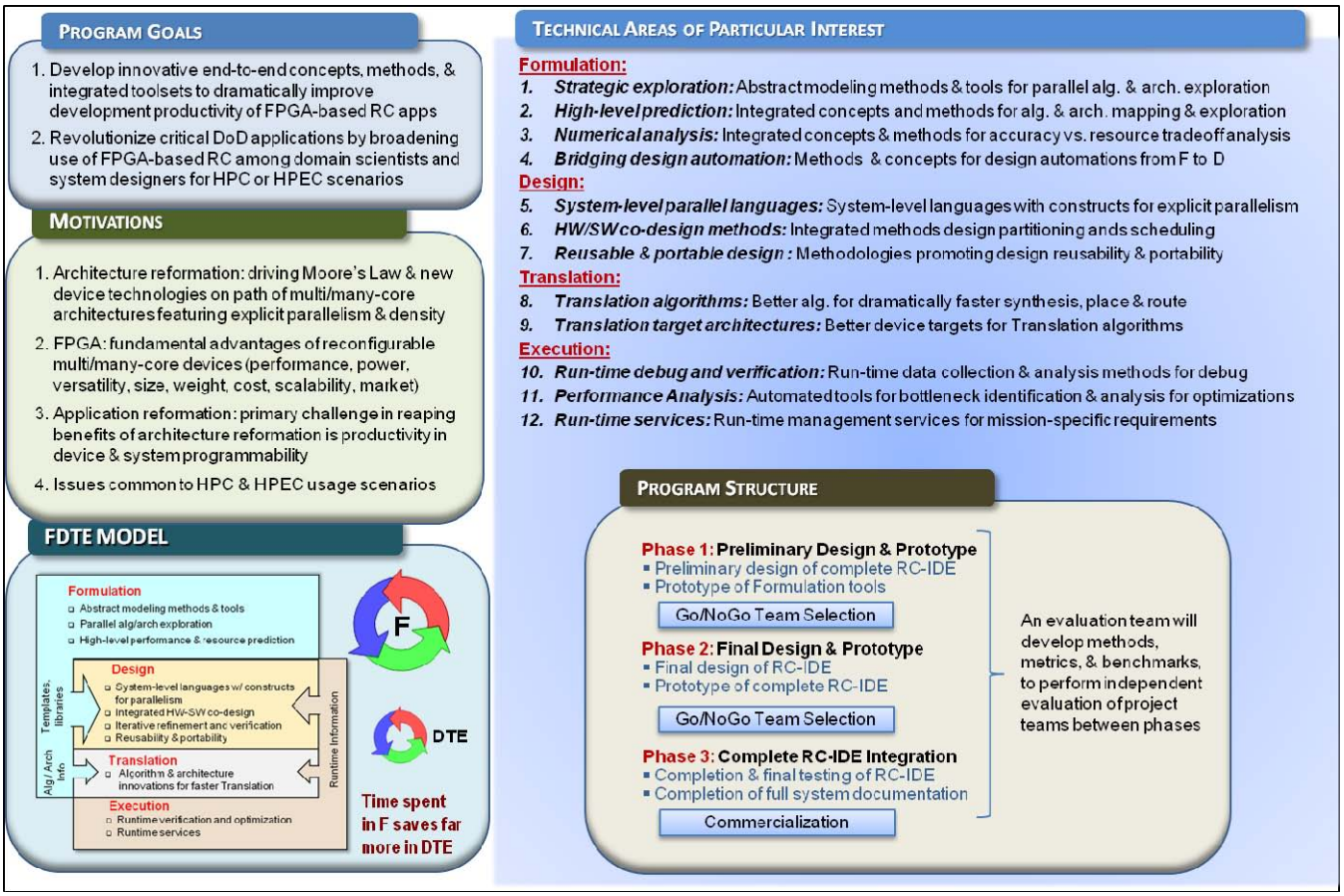


Figure 39. SIRCA program roadmap

7 Integrated Research Vision

During the past year, two independent studies have been performed, each charged with defining a vision and roadmap for addressing fundamental challenges in application development tools for FPGA-based systems. The outcomes from these two studies are described in two reports entitled Strategic Infrastructure for Reconfigurable Computing Applications (SIRCA) and FPGA Design Productivity (FDP). The purpose of this section is to describe an integrated research vision that includes the major concepts and research approaches from these two studies in a unified and integrated manner.

The two study teams met together on June 5th, 2008 in Salt Lake City along with experts in the field to present the results of their findings and begin the task of integrating the research vision presented by both teams. Breakout groups at the meeting provided feedback and suggestions on how to integrate the results from these research studies. We believe that this unified vision can form the basis of a potential new program that will lead to revolutionary improvements in design productivity for reconfigurable computing systems.

The two teams worked independently to query the reconfigurable computing community, gain a solid understanding of contemporary practices, and research past and current endeavors related to FPGA design productivity. Surprisingly, the two teams presented findings that shared several common themes. Both teams discussed similar causes to the problem and presented similar approaches for addressing the challenges in application development for FPGA-based systems. However, each team approached its study in a unique manner and emphasized different aspects of the design methodology. While the emphasis of each study was different, the results of both studies complement each other well and when taken together present a clear and complete research plan for significantly improving FPGA design productivity.

The SIRCA team organized its study around the concepts of Formulation, Design, Translation, and Execution (FDTE). This research model is defined horizontally in terms of the four fundamental stages in application development. The SIRCA study emphasizes research challenges in all four of these development stages but especially the Formulation stage, which features strategic design exploration and tradeoff analyses for complex systems and is pivotal for design productivity in many fields of engineering, and yet routinely overlooked in conventional hardware and software engineering.

The FDP team organized its study around three research focus areas: Abstraction, Reuse, and Turns per day (ART). This research model is defined vertically, where each research focus area defines a key research thrust that must be addressed in all stages of application development. The FDP study emphasizes the need to increase abstraction (reduce design detail), apply reuse, and reduce turns per day at all stages of the design process to obtain significant improvements in design productivity.

Figure 40 visually demonstrates the relationship between the models presented by the two study teams. In the center, application development is defined in terms of the four stages in the FDTE model. The process begins with Formulation, featuring strategic exploration of candidate algorithms and architectures supported by performance prediction for tradeoff analyses. After strategic decisions are made, the process moves to code design and implementation in the Design stage, then Translation to produce an executable form, and finally Execution, where verification and optimization occur and the application executes supported by a variety of run-time services. The arrows between stages emphasize the iterative nature of the development process and importance of exploiting results (templates, libraries, patterns, run-time information, etc.) between stages.

Each of the three research themes of the ART model are shown as vertical bars that span all development stages of the FDTE model. Reuse, for example, can be applied during Formulation, Design, Translation, and Execution to significantly reduce the amount of new work that must be performed by a programmer or by automated design tools. The other two focus areas, abstraction and turns per day, also span the four design stages of the FDTE model – technical approaches for each of these focus areas are possible at each design stage to improve programmer productivity.

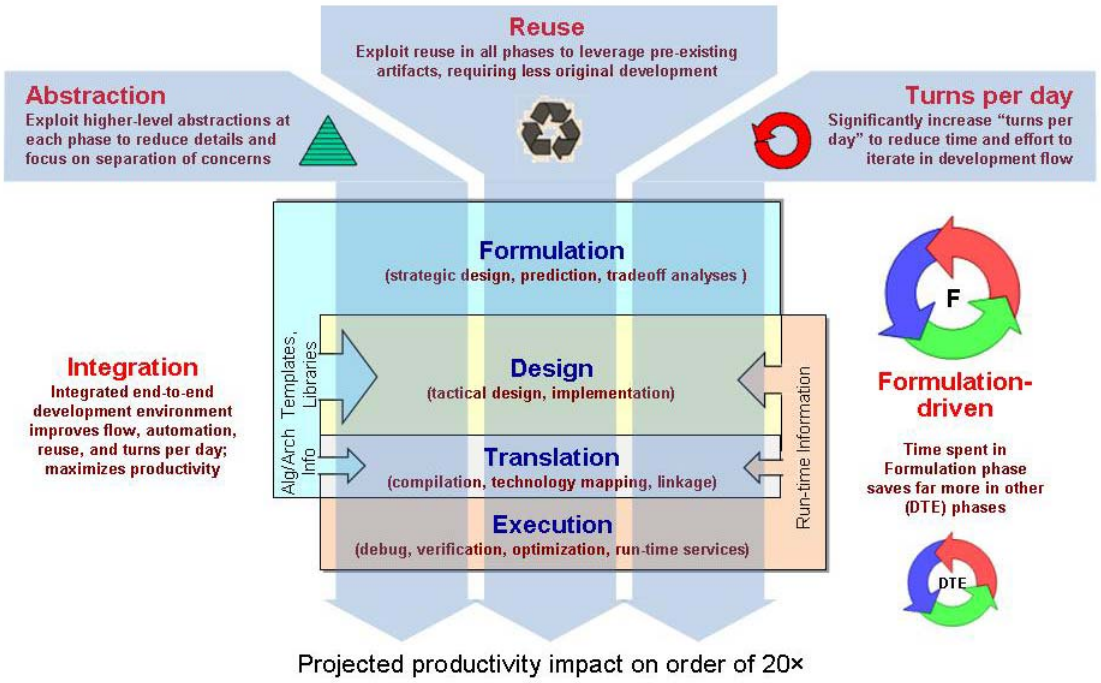


Figure 40. Integrated Research Vision

Each of the teams identified a set of specific research thrusts that will lead to major improvements in design productivity, 21 thrusts all tolled. As highlighted in Table 7, each of these research thrusts can be placed within the integrated research vision of Figure 40. The two study teams believe that improvements in design productivity of 20× or better are possible if advancements are made with each of the development stages of the FDTE model and focused in terms of abstraction, reuse, and turns per day.

8 Summary and Conclusions

The proposed SIRCA program is of vital importance for supporting future industrial, scientific and/or defense missions from satellites to supercomputers, and featuring advantages in speed, low-power, versatility, size, etc. However, the chief roadblock for these technologies is productivity in application development for RC systems.

A government led effort can drive innovation on this path in a coherent way. By contrast, the prevailing industry in this field is fragmented, populated by disparate types of companies (e.g. from large device vendors to small tools vendors), and focused primarily on incremental, evolutionary steps instead of revolutionary breakthroughs.

The proposed SIRCA program will set the foundation to meet these critical needs. Impact projections have shown the potential for a dramatic decrease in application development cost, up to 12× by conservative estimates and 16× by optimistic ones. This achievement would for example imply a reduction in development effort from one man-year to one man-month or less, which is a dramatic increase in productivity. When factoring in utility as well, where productivity is the ratio of utility versus development cost, the results are even more dramatic. Estimates suggest a gain in overall productivity of 30× and more can be achieved versus conventional HPC and HPEC systems that are based upon fixed-architecture multicore technologies. The ultimate outcome from these achievements when rendered is that SIRCA will invigorate widespread exploitation of RC acceleration for a wide spectrum of applications.

Table 7. Research Thrusts and Models

Thrusts	ART Model			FDTE Model			
	Abstraction	Reuse	Turns/day	Formulation	Design	Translation	Execution
SIRCA Research Thrusts							
1. Strategic exploration	X	X	X	X			
2. High-level prediction	X		X	X			
3. Numerical analysis	X	X		X			
4. Bridging design automation		X	X	X	X		
5. System-level parallel languages	X	X			X	X	
6. HW/SW codesign methods	X	X			X	X	
7. Reusable & portable design		X		X	X	X	
8. Translation algorithms			X			X	
9. Translation target architectures			X			X	
10. Run-time debug & verification	X		X		X		X
11. Performance analysis	X		X		X		X
12. Run-time services	X		X				X
FDP Research Thrusts							
1. Architecture shaping		X				X	X
2. Dual-layer compilation		X		X	X		
3. Libraries & standards		X			X	X	
4. Interface synthesis		X	X		X	X	
5. Parallel environments	X	X		X	X		
6. Multi-FPGA synthesis	X	X		X	X		
7. Platform services			X				X
8. Firmware			X			X	X
9. High-level debug	X		X				X

9 References

[AADL] Architecture Analysis & Design Language (AADL), http://www.aadl.info/.

[Agarwal-07] A. Agarwal, "The Tile processor: A 64-core multicore for embedded processing," *Proc. High-Performance Embedded Computing Workshop (HPEC)*, MIT Lincoln Lab, Lexington, MA, Sep. 18-20, 2007.

[Altera-99] "SignalTap User's Guide," *Altera Corporation*, San Jose, CA, 1999.10 (rev. 2) ed., Nov. 1999.

[Antola-07] A. Antola, "A Novel Hardware/Software Codesign Methodology Based on Dynamic Reconfiguration with Impulse C and Codeveloper," *Proc. Third Southern Conference on Programmable Logic (SPL '07)*, pp. 221-224, 2007.

[Ashby-07] J. V. Ashby, "New Languages for High Performance, High Productivity Computing," *RAL Technical Reports*, RAL-TR-2007-012, 2007.

[Balarin-97] F. Balarin, M. Chiodo, P. Giusto, H. Hsieh, A, Jurecska, L. Lavagno, C. Passerone, A. Sangiovanni-Vincentelli, E. Sentovich, K. Suzuki, B. Tabbara, "Hardware-Software Co-Design of Embedded Systems: The Polis Approach," *Kluwer Academic Press*, Jun. 1997.

[Balarin-03] F. Balarin, "Metropolis: an integrated electronic system design environment," *Computer*, Vol. 36, pp. 45-52, 2003.

[Bellows] P. Bellows and B. Hutchings, "JHDL - An HDL for Reconfigurable Systems," http://www.jhdl.org/papers.html.

[Betz-97] V. Betz and J. Rose, "VPR: a new packing, placement and routing tool for FPGA research", *Proc. Seventh International Workshop on Field-Programmable Logic and Applications (FPL97)*, pp 213-222, London, UK, Sept 1-3, 1997.

[Buttari-07] A. Buttari, J. Dongarra, J. Langou, J. Langou, P. Luszczek, and J. Kurzak, "Mixed Precision Iterative Refinement Techniques for the Solution of Dense Linear Systems, *International Journal of High Performance Computer Applications*, Vol. 21, No. 4, pp. 457-466, Aug. 2007.

[Buyukkurt-06] B. Buyukkurt, Z. Guo, and W. Najjar, "Impact of Loop Unrolling on Throughput, Area and Clock Frequency in ROCCC: C to VHDL Compiler for FPGAs," *Proc. International Workshop On Applied Reconfigurable Computing (ARC)*, Delft, The Netherlands, Mar. 1-3, 2006.

[Camera-05] K. Camera, H. K. So and R. W. Brodersen, "An integrated debugging environment for reprogrammble hardware systems," *Proc. of the Sixth International Symposium on Automated Analysis-Driven Debugging (AADEBUG)*, Monterey, California, pp. 111-116, 2005.

[Compton-02] K. Compton and S. Hauck, "Reconfigurable computing: A survey of systems and software," *ACM Computing Surveys (CSUR)*, Vol. 34, No. 2, pp. 171-210, Jun. 2002.

[Curreri-08] J. Curreri, S. Koehler, B. Holland, and A. George, "Performance Analysis with High-Level Languages for High-Performance Reconfigurable Computing," *Proc. of 16th IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM)*, Palo Alto, CA, Apr. 14-15, 2008.

[Dehon-04] A. DeHon, J. Adams, M. DeLorimier, N. Kapre, Y. Matsuda, H. Naeimi, M. Vanier, and M. Wrighton, "Design patterns for reconfigurable computing," *Proc. Twelfth Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pp. 13-23, Napa Valley, CA, Apr. 20-23, 2004.

[Densmore-06] D. Densmore, A. Sangiovanni-Vincentelli, and R. Passerone, "A Platform-Based Taxonomy for ESL Design", *IEEE Design & Test of Computers*, Volume 23, Issue 5, pp. 359-374, May 2006.

[DSPB] Altera DSP Builder, http://www.altera.com/products/software/products/dsp/dsp-builder.html

[Edwards-97] S. Edwards, L. Lavagno, E.A. Lee, and A. Sangiovanni-Vincentelli, "Design of Embedded Systems: Formal Models, Validation, and Synthesis", *Proc. IEEE*, Volume 85, Issue 3, March 1997.

[El-Araby-07a] E. El-Araby, P. Nosum, and T. El-Ghazawi, "Productivity of High-Level Languages on Reconfigurable Computers: An HPC Perspective", *Proc. IEEE International Conference on Field-Programmable Technology (FPT 2007)*, Japan, December, 2007.

[El-Araby-07b] E. El-Araby, M. Taher, M. Abouellail, T. El-Ghazawi, and G. B. Newby, "Comparative Analysis of High Level Programming for Reconfigurable Computers: Methodology and Empirical Study", *Proc. Third Southern Conference on Programmable Logic (SPL2007)*, Mar del Plata, Argentina, February, 2007.

[Fobel-07] C. Fobel, G. Grewal, A. Morton, "Using Hardware Acceleration to Reduce FPGA Placement Times," *Proc. Canadian Conference on Electrical and Computer Engineering (CCECE)*, pp. 647-650, Vancouver, Canada, Apr. 22-26, 2007.

[Fornaciari -02] W. Fornaciari, D. Sciuto, C. Silvano, V. Zaccaria, 'A sensitivity-based design space exploration methodology for embedded systems," *Design Automation for Embedded Systems, Kluwer Academic Publishers* 7 (1-2), pp. 7-33, 2002.

[Gamma-94] E. Gamma, R. Johnson, H. Helm, J. M. Vlissides, and G. Booch, "Design Patterns: Elements of Reusable Object-Oriented Software", *Addison-Wesley Professional*, 1994.

[Gedae] Gedae, www.gedae.com

[Gries-03] M. Gries, "Methods for Evaluating and Covering the Design Space during Early Design Development", *Technical report, Electronics Research Lab, University of California at Berkeley*, UCB/ERL M03/32pp. 189-194, August 2003.

[George-08] A. George, "Reformation and Formulation: Key Challenges for Reconfigurable Supercomputing," *Proc. of Many-core and Reconfigurable Supercomputing Conference (MRSC)*, Belfast, Ireland, Apr. 1-3, 2008 (keynote presentation).

[Graham-01] P. Graham, B. Nelson, and B. Hutchings, "Instrumenting Bitstreams for Debugging FPGA Circuits," *Proc. of Ninth IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM'01)*, pp. 41-50, Apr. 29 - May 2, 2001.

[Grobelny-07] E. Grobelny , C. Reardon, A. Jacobs, and A. George, "Simulation Framework for Performance Prediction in the Engineering of RC Systems and Applications," *Proc. International Conference on Engineering of Reconfigurable Systems and Algorithms (ERSA)*, Las Vegas, NV, Jun. 25-28, 2007.

[Gruian-06] F. Gruian, "The SystemJ approach to system-level design," *Proc. Fourth ACM and IEEE International Conference on Formal Methods and Models for Co-Design (MEMOCODE '06)*, pp. 149-158, Jul 2006.

[Haldar-00] M. Haldar, A. Nayak, A. Choudhary, P. Banerjee "Parallel algorithms for FPGA placement," *Proc. Tenth Great Lakes symposium on VLSI*, pp. 86-94, Chicago, IL, Mar. 2-4, 2000.

[Holland-07] B. Holland, K. Nagarajan, C. Conger, A. Jacobs, and A. George, "RAT: A Methodology for Predicting Performance in Application Design Migration to FPGAs," *Proc. High-Performance Reconfigurable Computing Technologies and Applications Workshop (HPRCTA)* at SC'07, Reno, NV, Nov. 11, 2007.

[Huang-07] Miaoqing Huang, Ivan Gonzalez, and Tarek El-Ghazawi, "A Portable Memory Access Framework for High-Performance Reconfigurable Computers", *Proc. IEEE International Conference on Field-Programmable Technology (ICFPT'07)*, Kokurakita, Kitakyushu, Japan, Dec. 12-14, 2007

[Huang-08] M. Huang, I. Gonzalez, S. Lopez-Buedo, and T. El-Ghazawi, "A Framework to Improve IP Portability on Reconfigurable Computers," to appear *Proc. 2008 International Conference on Engineering of Reconfigurable Systems and Algorithms (ERSA)*, Las Vegas, NV, July 14-17, 2008.

[ImpulseC] Impulse Accelerated Technologies, "Accelerating HPC and HPEC Applications Using Impulse C", Reconfigurable Systems Summer Institute (RSSI), Urbana, IL, July 17-20, 2007

[Kannan-06] P. Kannan, D. Bhatia, "Interconnect estimation for FPGAs," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on* , vol.25, no.8, pp. 1523-1534, Aug. 2006.

[Kendall-05] R. Kendall et al., "Case Study of the Hawk Code Project," *LLANL technical report* # LA-UR-05-9011, 2005.

[Kepner-04a] J. Kepner, "HPC Productivity: An Overarching View," *Int. Journal of High Performance Computing Applications*, Vol 18, No. 4, pp. 393-397, 2004.

[Kepner-04b] J. Kepner, "High Performance Computing Productivity Model Synthesis," *Int. Journal of High Performance Computing Applications*, Vol 18, No. 4, pp. 505-516, 2004.

[Koehler-07] S. Koehler, J. Curreri, and A. George, "Challenges for Performance Analysis in High-Performance Reconfigurable Computing," *Proc. Reconfigurable Systems Summer Institute (RSSI)*, Urbana, IL, Jul. 17-20, 2007.

[LabView] LabView, http://www.ni.com/labview/

[Lee-06] D. Lee, A. Abdul Gaffar, R.C.C. Cheung, O. Mencer, W. Luk, and G.A. Constantinides, "Accuracy Guaranteed Bit-Width Optimization", *IEEE Transactions on Computer-Aided Design*, Vol. 25(10), 1990-2000, Oct. 2006.

[Li-95] J. Li, C. Cheng, "Routability improvement using dynamic interconnect architecture", *Proc. IEEE Symposium on FPGAs for Custom Computing Machines*, pp. 61-67, 19-21 April 1995.

[Li-06] X. Li, H. Yang, and H. Zhong; "Use of VPR in Design of FPGA Architecture", *Proc. Eight International Conference on Solid-State and Integrated Circuit Technology, ICSICT '06)*, pp.1880 – 1882, 2006.

[Lubbers-07] E. Lubbers and M. Platzner, "RECONOS: An RTOS Supporting Hard- and Software Threads," *Proc. Field Programmable Logic and Applications (FPL)*, Amsterdam, the Netherlands, Aug. 27-29, 2007.

[LViewFPGA] LabVIEW FPGA, http://www.ni.com/fpga/

[Lysecky-04] R. Lysecky, F. Vahid, and S. Tan, "Dynamic FPGA Routing for Just-in-Time Compilation," *IEEE/ACM Design Automation Conference (DAC)*, June 2004.

[Lysecky-06] R. Lysecky, G. Stitt, and F. Vahid, "Warp Processors," *ACM Trans. Design Automation Electronic Systems (TODAES)*, Vol. 11, pp. 659-681, Jul. 2006.

[Masuno-07] S. Masuno, T. Maruyama, Y. Yamaguchi, and A. Konagaya, "Multiple Sequence Alignment Based on Dynamic Programming Using FPGA," Proc. *IEICE Trans. Information and Systems*, Vol. E90-D, No. 12, Dec 2007.

[Merchant-07] S. Merchant, "Intrinsically Evolvable Artificial Neural Networks," *Ph.D. dissertation, ECE Dept. University of Tennessee*, Aug 2007.

[Mulpuri-01] C. Mulpuri and S. Hauck, "Runtime and quality tradeoffs in FPGA placement and routing," *Proc. ACM/SIGDA Ninth International Symposium on Field Programmable Gate Arrays*, pp. 29-36, Monterey, CA, 2001.

[Nagarajan-08] K. Nagarajan, B. Holland, A. George, C. Slatton, H. Lam, "Accelerating Principal Machine-Learning Algorithms on FPGAs with a Pattern-based Design Methodology," submitted and pending review *Journal of Signal Processing Systems*, 2008.

[OpenFPGA] OpenFPGA, http://openfpga.org.

[Planti-99] J. Plantin, "Aspects on system-level design," *Hardware/Software Codesign (CODES '99), Proc. Seventh International Workshop on*, pp. 209-210, 1999.

[Pellerin-05] D. Pellerin and S. Thibault, "Practical FPGA Programming in C," *Prentice Hall PTR*, 2005.

[Penttinen-06] A. Penttinen, R. Jastrzebski, R. Pollanen, O. Pyrhonen, "Run-time Debugging and Monitoring of FPGA Circuits Using Embedded Microprocessors," *Proc. IEEE Design and Diagnostics of electronic Circuits and Systems*, Prague, Czech Republic, Apr. 18-21, 2006.

[Poznanovic-05] D. S. Poznanovic, "Application Development on the SRC Computers, Inc. Systems," *Proc. Nineteenth IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, Denver, CO, pp. 78a, Apr 2007.

[Saha-07a] P. Saha and T. El-Ghazawi, "Applications of Heterogeneous Computing in Hardware/Software Co-Scheduling," *Proc. ACS/IEEE International Conference on Computer Systems and Applications*, May 13-16, 2007.

[Saha-07b] P. Saha and T. El-Ghazawi, "A Methodology for Automating Co-Scheduling for Reconfigurable Computing Systems," *Proc. Fifth ACM/IEEE International Conference on Formal Methods and Models for Codesign*, May 30-Jun 1 2007.

[Sangiovanni-Vincentelli-07] A. Sangiovanni-Vincentelli, "Quo Vadis, SLD? Reasoning About the Trends and Challenges of System-Level Design," *Proc. IEEE*, Vol. 95, pp. 467-506, Mar. 2007.

[Scrofano-07] R. Scrofano and V. Prasanna, “A Hierarchical Performance Model for Reconfigurable Computers,” in “Handbook of Parallel Computing: Models, Algorithms and Applications,” edited by S. Rajasekaran and J. Reif, *CRC Press*, 2007.

[Shackleford-01] B. Shackleford, G. Snider, R. Carter, E. Okushi, M. Yasuda, K. Seo, and H. Yasuura “A High-Performance, Pipelined, FPGA-Based Genetic Algorithm Machine,” *Genetic Programming and Evolvable Machines*, Vol. 2, pp. 33-60, 2001

[Simulink] Simulink - Simulation and model-based design, http://www.mathworks.com/products/simulink/

[Smith-03] M.C. Smith, "Analytical modeling of high performance reconfigurable computers: prediction and analysis of system performance," *Ph.D. Dissertation, ECE Dept. University of Tennessee*, pp. 208, Dec. 2003.

[Stitt-07] G. Stitt and F. Vahid, "Thread warping: a framework for dynamic synthesis of thread accelerators," *Proc. International Conf. on Hardware/Software Co-design and System Synthesis (CODES/ISSS)*, pp. 93-98, Salzburg, Austria, Sept. 30-Oct. 5, 2007.

[Sysgen] Xilinx System Generator, http://www.xilinx.com/ise/optional_prod/system_generator.htm

[SystemC] “Open SystemC initiative,” www.systemc.org.

[T-APPLIB] T-APPLIB: Technical working group on application specific FPGA libraries, *OpenFPGA*, http://openfpga.org/.

[T-CORELIB] T-CORELIB: Technical working group on core libraries and interoperability, *OpenFPGA*, http://openfpga.org/.

[T-GENAPI] T-GENAPI: Technical working group on general FPGA functionality APIs, *OpenFPGA*, http://openfpga.org/.

[Tomko-00] K. Tomko and A. Tiwari, "Hardware/software Co-debugging for Reconfigurable Computing," *Proc. of IEEE International High-Level Design Validation and Test Workshop*, pp.59-63, Berkeley, CA, Nov. 8-10, 2000.

[Tong-07] J. G. Tong, "A Comparison of Profiling Tools for FPGA-Based Embedded Systems", *Electrical and Computer Engineering, CCECE 2007, Canadian Conference on*, pp. 1687-1690, 2007.

[Troxel-06] I. Troxel, E. Grobelny, G. Cieslewski, J. Curreri, M. Fischer, and A. George, "Reliable Management Services for COTS-based Space Systems and Applications," *Proc. International Conference on Embedded Systems and Applications (ESA)*, Las Vegas, NV, Jun. 26-29, 2006.

[UML] Object Management Group: Unified Modeling Language (UML), http://www.uml.org/.

[Wang-96] P. Wang, and K. Chen, "A simultaneous placement and global routing algorithm for an FGPA with hierarchical interconnection structure", Proc. *IEEE International Symposium on Circuits and Systems (ISCAS'96)*, Vol.4, pp. 659-662, May 1996.

[Williams-08a] J. Williams, A. George, J. Richardson, K. Gosrani, and S. Suresh, "Computational Density of Fixed and Reconfigurable Multi-Core Devices for Application Acceleration," accepted and to appear in Proc. Reconfigurable Systems Summer Institute 2008 (RSSI), Urbana, IL, July 7-10, 2008.

[Williams-08b] J. Williams, A. George, J. Richardson, K. Gosrani, and S. Suresh, "Fixed and reconfigurable multi-core device characterization for HPEC," submitted and review pending in Proc. High-Performance Embedded Computing Workshop (HPEC), MIT Lincoln Lab, Lexington, MA, Sep. 23-25, 2008.

[Wrighton-03] M. Wrighton and A. DeHon, "Hardware-assisted simulated annealing with application for fast FPGA placement,“ *Proc. International Symposium on Field-Programmable Gate Arrays (FPGA)* pp.33-42, Feb. 23-25, 2003.

[Writhlin-07] M. Wirthlin, D. Poznanovic, P. Sundararajan, A. Coppola, D. Pellerin, W. Najjar, R. Bruce, M. Babst, O. Pritchard, P. Palazzari1, and G. Kuzmanov, “OpenFPGA CoreLib Core Library Interoperability Effort,” *Proc. Reconfigurable Systems Summer Institute (RSSI)*, Urbana, IL, Jul. 17-20, 2007.

[Xilinx-00] “ChipScope Software and ILA Cores User Manual,’ *Xilinx*, San Jose, CA, v. 1.1 ed., Jun. 2000.